UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Asistente para consultas espaciales en Hive

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

AUTOR: José Emilio Maldonado Muñoz

TUTOR: Dolores Cuadra Fernández

Junio de 2015

Agradecimientos

Con este proyecto pongo fin a una etapa espectacular. Durante estos seis años en total he vivido experiencias buenas y malas, pero sobre todo me quedo con los compañeros y amigos que me llevo. Compañeros con los que he compartido aprobados, suspensos, esperas de notas (esos 5.0 poderosos), sábados de madrugada acabando prácticas, domingos por la noche entregando de última hora, días de césped y 'birreo'… Y risas, muchas risas. Sólo por eso ha merecido la pena. En especial hago mención a Epi, Sergio y Dani, grandes colegas más allá de haber sido compañeros.

Por otro lado quiero agradecer el apoyo recibido por parte de mis padres y hermano, tanto para acabar las últimas asignaturas como para dar el último paso con este proyecto. No sólo a ellos, si no a mis abuelos, y en especial a mi abuelo Antonio, que recientemente se nos fue. Este título es para ellos.

Y no me puedo olvidar de mi apoyo incondicional, Patricia. Gracias por aguantarme tantos lloros por suspensos y asignaturas imposibles.

Por último agradecer a mi tutora Dolores el apoyo durante este año para acabar el proyecto, así como a Harith por la ayuda adicional.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1. Introducción

Para entender la finalidad y la utilidad del caso práctico que aquí se desarrolla es necesario comprender las motivaciones que han originado el trabajo, así como los objetivos que se han marcado a partir de dichas motivaciones.

Por otro lado, para facilitar la lectura y comprensión del documento se proporciona un resumen general del trabajo y un índice de la estructura de la memoria y el cometido de cada apartado.

1.1. Resumen

El trabajo descrito y desarrollado a lo largo de esta memoria tiene como objetivo principal establecer un esquema de decisiones básicas para realizar consultas sobre datos espaciales en Hadoop/Hive. Dicho esquema servirá de asistente y soporte para usuarios nuevos en Hive, que teniendo nociones mínimamente básicas sobre el lenguaje de consultas estándar SQL, quieran trabajar en un ámbito Big Data sobre datos espaciales. En definitiva, este trabajo sirve para facilitar y asistir la primera toma de contacto con esta tecnología.

Con este fin, se ha realizado un estudio teórico inicial sobre el ámbito y contexto de trabajo (tendencias, tecnologías, etc.); se ha diseñado e implementado un caso práctico sobre el que se ha realizado la experimentación; y por último, se ha construido un árbol genérico de decisiones, donde sobre cada rama del mismo se han seleccionado, explicado y probado ejemplos destacados de la batería masiva de consultas experimentales fruto de la experimentación en Hive. Dicha batería de consultas se ha realizado partiendo de patrones de diseño identificados en un Trabajo Fin de Grado anterior. La conclusión principal de este trabajo debe estar encaminada a corroborar que dichos patrones son la base de construcción de consultas espaciales en Hive.

Por último, este trabajo también pretende ser la base para la investigación acerca de los datos espaciales dentro de un entorno Big Data. Al igual que este proyecto toma de base trabajos anteriores, éste servirá como pieza inicial en futuros estudios encaminados en esta línea.

1.2. Motivación y objetivos

Actualmente cruzamos la era de la información. Internet ha propiciado la compartición masiva de datos de todo tipo: financieros, personales, noticias, operacionales, etc. Nunca antes se había tenido acceso a tal volumen de información. Pero no sólo es la era de la información, sino que también es la era del consumo. Esos dos conceptos, información y consumo, hacen que las partes interesadas en colocar sus productos en un mercado voraz lleno de competitividad estén obligadas a conocer a la perfección las tendencias escondidas entre tantos datos y a manejarse de forma rápida entre ellos. Eso ha sido así hasta ahora, pero en ocasiones, como se verá más adelante, ya no vale con manejar la información con los cánones clásicos de almacenamiento, sino que es preciso el procesamiento de muchísimos más datos y hacerlo en el menor tiempo posible.

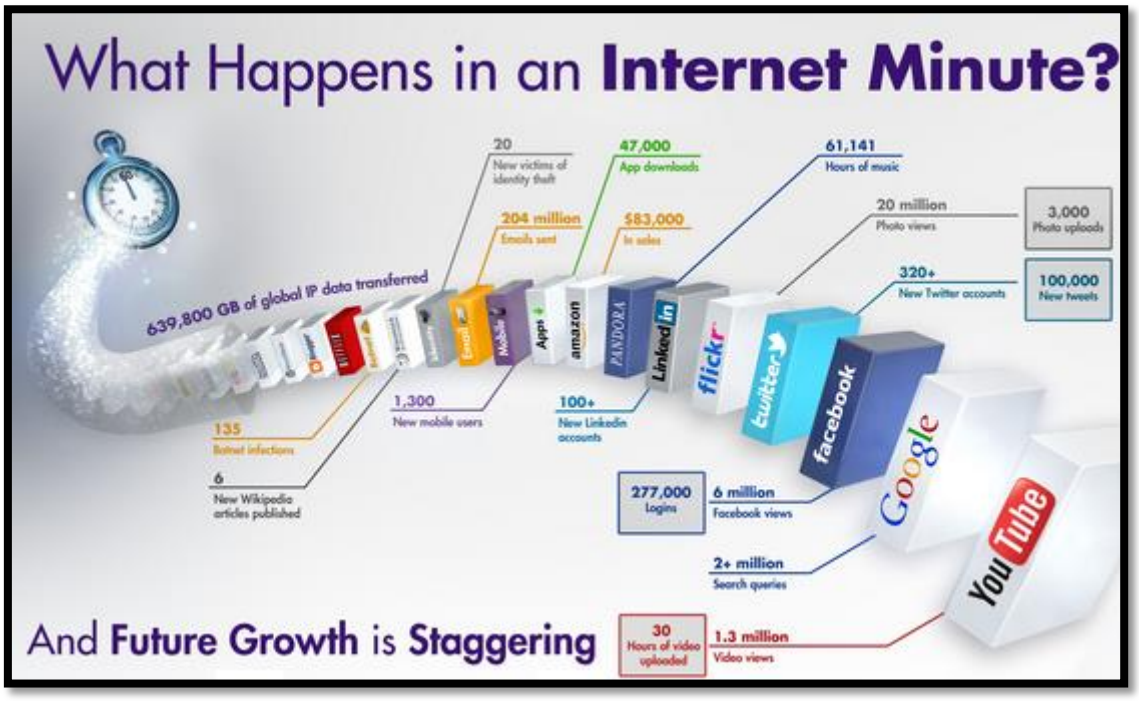

Ilustración 1: Tráfico de información en la red producido por las principales webs

URL: http://pixelsandpills.com/2013/03/27/tictrac-a-digital-health-data-aggregation-tool/

La mayoría de sistemas informáticos que comparten información a través de Internet han estado utilizando casi en su totalidad esos cánones clásicos para almacenar datos. Dichos cánones clásicos se corresponden, en gran medida, con las llamadas **bases de datos basadas en transacciones**, las cuales tienen su principal ejemplo en las bases de datos relacionales, construidas a partir del *Standard Query Laguage* (SQL). Pero el cada vez mayor volumen de información (Ilustración 1) manejado y distribuido por determinados tipos de sistemas ha propiciado la aparición de los conceptos No SQL (*2.2. Bases de datos relacionales y no relacionales. SQL y NO-SQL*) y Big Data (*2.1. Big Data*), no sólo por su significado literal (No Standard, Big → grande) los cuáles se desmarcan de los estándares, sino por la forma en que la información es tratada en ellos. Como se explicará más adelante en este trabajo, las necesidades de un tipo u otro de almacenamiento varían, y es preciso saber cuándo es conveniente usar unas u otras tecnologías.

Este trabajo arroja luz sobre las tecnologías y herramientas para manejar grandes volúmenes de datos en un entorno no relacional. La particularidad del proyecto es que se trabaja sobre **datos de naturaleza geográfica**, los cuales están muy bien soportados por estándares en entornos relacionales, pero que en entornos no relacionales no siempre es así. Por ello este trabajo adquiere su principal hándicap en **proporcionar una guía de introducción para trabajar sobre una plataforma Big Data, con el añadido de operar sobre datos espaciales**, siendo capaz el lector de entender además cómo **realizar diseños de una consulta espacial teniendo en cuenta patrones de diseño.** Con los resultados de este proyecto no sólo se facilita el acceso

a estas tecnologías, sino que **sirve de base y análisis para futuros proyectos, tanto de investigación como de desarrollo de aplicaciones con base Hadoop.**

En resumen, las principales motivaciones sobre las que se construye el trabajo son:

- **Introducir al conocimiento de las tecnologías utilizadas dentro del ámbito Big Data.** Dichas tecnologías son muy recientes, y son cada vez usadas con más frecuencias en el ámbito de la empresa privada. El foco de atención estará puesto en la explotación de datos espaciales.

- **Poca documentación y ejemplos prácticos de implantación y uso de estas tecnologías.** El uso de estas tecnologías es muy nuevo y no existe apenas documentación accesible para iniciarse en ellas. Pocas corporaciones como Hortonworks o Datastax proporcionan manuales o guías básicas sobre la utilización e instalación de entornos Big Data, y menos aún existen fuentes donde consultar posibles soluciones a problemas particulares con estas herramientas.

- **Uso y explotación de datos geográficos en ámbito de Big Data.** El estándar PostGIS (*2.3.1. PostGIS y GIS Tools Hadoop*) proporciona una serie de bibliotecas espaciales para tratar datos de índole geográfico en entornos relacionales, pero en ocasiones es complejo de realizar esta tarea en entorno Big Data. El caso práctico por lo tanto se especializa en el manejo de este tipo de datos, y busca encontrar soluciones en un entorno Big Data, los cuales son triviales en entorno relacional.

- **Diseño de consultas espaciales a partir de patrones definidos en un proyecto anterior.** Se proporcionará un conjunto de directrices para resolver problemas de índole espacial. Es interesante agrupar ejemplos de esta tecnología justo cuando ésta empieza a ser cada vez más utilizada.

Como resultado de las principales motivaciones, se establecen los siguientes objetivos con el fin de resolver la problemática planteada:

- Ampliar conocimientos de tecnología incipiente, tanto a nivel teórico como práctico. En este documento se reflejará parte de este estudio en el estado del arte y parte en la documentación del trabajo realizado.

- Partir de *data sets* espaciales y de patrones de diseño de consultas ya obtenidos en un proyecto anterior a este.

- A partir de los *data sets*, diseñar un caso práctico sobre el que trabajar con los patrones.

- Diseñar, implementar y probar un número suficiente de consultas espaciales como para estudiar la validez de los patrones.

- Demostrar mediante la construcción de un esquema de diseño a modo de asistente que el conjunto de consultas diseñadas, programadas y probadas demuestran por sí solas que los patrones de diseño utilizados son válidos para construir búsquedas sobre datos espaciales. Este ejercicio demostrativo pretende ser la base de futuros trabajos que sigan la misma línea que éste, ya sea en el campo del estudio teórico, caso práctico o proyecto de desarrollo.

1.3. Estructura del documento

El documento recoge todo el trabajo realizado tanto en el caso práctico como en la investigación previa sobre el contexto y sobre los entornos de desarrollo. Para facilitar su lectura se ha estructurado éste en un total de nueve apartados principales bien definidos (más anexos) y con contenidos diferenciados. Son los siguientes:

- 1. Introducción: Presentación de las principales motivaciones que han dado como fruto la realización de este trabajo, y los objetivos que se han establecido a raíz de esas motivaciones. Resumen general del trabajo realizado y estructuración del documento.

- 2. Estado del arte: Contextualización del problema planteado dentro de su marco tecnológico y a nivel de tendencias. Justificación por medio de su contexto de la importancia que adquiere el trabajo para cubrir una necesidad específica.

- 3. Descripción del caso práctico: Eje central del documento. Se detallarán todo el proceso de trabajo en el caso práctico, desde la concepción inicial, manejo de estructuras y datos, hasta la elaboración de consultas espaciales, clasificación de las mismas y establecimiento de resultados probados a partir ellas.

- 6. Conclusiones y futuras líneas de trabajo: Crítica de la calidad de los resultados obtenidos y de si estos han cumplido las expectativas iniciales. Presentación de futuras líneas de trabajo, líneas que hayan surgido a lo largo del desarrollo del proyecto y puedan servir para futuros trabajos.

- 7. Aspectos legales: Resumen de las consideraciones a nivel de licencias y derechos de autor de las librerías y aplicaciones de las que se ha hecho uso en el proyecto, y justificación del uso de dichas herramientas según su componente legal.

- 8. Gestión del proyecto: Planificación del trabajo realizado, mostrando etapas y puntos de mayor carga de trabajo y comparando la estimación inicial de tiempos con la final. Por otra parte, análisis del presupuesto y costes del proyecto.

- 9. Bibliografía: Apartado donde aparecen reflejadas de manera ordenada y con criterio las distintas fuentes utilizadas como referencias en este trabajo. Se clasifican en dos grupo: aquellas cuyo contenido ha sido directamente incluido en el trabajo, y aquellas que han servido de consulta o que sirven para que el lector amplíe por su cuenta los

conocimientos acerca del tema tratado. Se aplicará el estándar IEEE de referenciación bibliográfica.

- Anexos: Información adicional sobre el proyecto. Contiene la configuración de entornos, scripts de configuración e implementación del caso práctico y listas de acrónimos y definición de términos técnicos o nombres de corporaciones que aparecen a lo largo de la memoria.

2. Estado del arte

La realización de este caso práctico no es casualidad. Las tecnologías que soportan Big Data son cada vez más utilizadas en el ámbito empresarial, no tanto dentro del territorio nacional, donde sólo grandes empresas hacen uso de ella, pero sí a escala mundial. No obstante, existe muy poca documentación accesible para el gran público tanto de las plataformas de trabajo así como de los lenguajes que soportan dichas plataformas, siendo muy escasos ejemplos de casos prácticos para iniciados.

Como se ha explicado en el punto *1.2. Motivación y objetivos*, este trabajo no sólo refleja una solución a un problema planteado, el establecimiento y verificación de patrones de diseño de consultas espaciales, sino que su objetivo también es proporcionar una vía de acceso al universo Big Data, enmarcado en este caso en los datos espaciales. En definitiva, el contexto del trabajo hace que adquiera importancia enmarcar éste dentro de su contexto, en su marco intelectual y tecnológico. Por ello, en este punto se explicarán los conceptos más importantes que rodean a las distintas tecnologías y paradigmas de datos que se han utilizado para la realización de este proyecto. Dichos conceptos son los siguientes:

- **Big Data:** Qué es, por qué surge y para qué uso está enfocado.
- **Bases de datos relacionales y no relacionales**: SQL y NoSQL. Principales paradigmas No SQL. Comparación de modelos.
- **Bases de datos espaciales y datos/funciones espaciales:** Qué son y para qué se usan, prestando especial atención a los sistemas de almacenamiento utilizados en este proyecto.
- **Hadoop/Hive y HQL:** Introducción a la herramienta, principales características y fundamentos de *Hive Query Language*.

2.1. Big Data

No puede entenderse el uso de bases de datos no relacionales sin comprenderse antes el concepto de Big Data. La misma expresión da una idea acerca de su significado: Big Data → "Muchos datos". Pese a ello, el concepto de Big Data, como cualquier otro que surge para definir algo novedoso, es algo abstracto. No hay una manera única de definirlo. ¿Qué significado tiene este término dentro del marco de las tecnologías de la información y el almacenamiento de datos? Para entenderlo, se aportan tres ejemplos de definiciones dadas por IBM, Microsoft y Oracle, empresas punteras en tecnologías de la información.

Ricardo Barranco de IBM describe este término de manera sencilla, poniendo énfasis en el cambio de tendencia actual en la forma de tratar la información:

"Tendencia en el avance de la tecnología que ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones, la cual es utilizada para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y

sería muy costoso cargarlos a un base de datos relacional para su análisis. De tal manera que, el concepto de Big Data aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales" [1].

Microsoft por su parte explica el concepto de Big Data entendiendo que el objetivo es interpretar conocimiento a partir de datos masivos aprovechando las nuevas tecnologías:

"*Big Data is the term increasingly used to describe the process of applying serious computing power – the latest in machine learning and artificial intelligence – to seriously massive and often highly complex sets of information*" [2].

Oracle define el término de manera más sencilla y genérica, haciendo hincapié en tres propiedades básicas que debe cumplir el Big Data:

"Amounts of information that are too difficult for traditional systems to handle, either the volume is too much, the velocity is too fast, or the variety is too complex" [3].

Hay más definiciones de este movimiento enunciadas por otros organismos o personalidades, pero en resumen todos coinciden en dejar patente que el movimiento Big Data **surge de la necesidad de resolver operaciones sobre grandes cantidades de datos de manera distinta a como se hacía hasta ahora, ya que existen restricciones que rigen sobre los gestores de bases de datos basadas en transacciones tradicionales (como las relacionales) las cuales ralentizan las distintas acciones que se van solicitando y no dan pie organizar la información de forma libre**.

Pese a que no hay un reglamento oficial que guíe o marque las directrices de diseño de un sistema enfocado al Big Data, existen tres propiedades comunes que la inmensa mayoría de profesionales dedicados a la explotación de datos aceptan como válidas. Estas tres propiedades, las conocidas como las 3V del Big Data, son las que citaba Oracle en su definición de este movimiento, y son las siguientes [4]:

- **Volumen:** Como ya se ha comentado, hay varios factores que han propiciado el incremento del flujo de datos. Pero es el surgimiento de las redes sociales y los dispositivos móviles, como los *smarthphones*, los que han hecho que ese flujo de información sea cada vez mayor. En el milenio pasado, el hardware era costoso de fabricar. Pero en las últimas dos décadas, conseguir máquinas potentes para procesar más y más información no es un problema mayor. Debido a esto, la atención ahora está puesta en cómo determinar qué datos son relevantes de entre todo ese *data set* masivo de información, y cómo crear valor de esos datos a partir de su análisis.

  En relación con el volumen de información, la empresa Cisco, líder en el mercado de las redes y las telecomunicaciones, vaticinaba en un estudio [5] que entre el 2011 y el 2016 la cantidad de tráfico de datos móviles crecería a una tasa anual de 78%, así como el número de dispositivos móviles conectados a Internet excedería el número de habitantes en el planeta. Así mismo, recogía el estudio que las naciones unidas proyectaban que la población mundial alcanzaría los 7.5 billones para el 2016 de tal modo que habría cerca de 18.9 billones de dispositivos conectados a la red a escala mundial, lo cual conllevaría a que el tráfico global de datos móviles alcanzase 10.8 Exabytes mensuales o 130 Exabytes anuales. Este volumen de tráfico previsto para 2016 equivale a 33 billones de DVDs anuales o 813 cuatrillones de mensajes de texto.

- **Velocidad:** Los datos fluyen a través de la red de manera muy rápida. Las grandes empresas informáticas están muy interesadas en que se procesen, muchas veces en tiempo real y con el menor retardo posible, los datos generados por sus aplicaciones. En definitiva, se necesita cubrir el desfase que existe entre el cada vez mayor flujo de datos y la rapidez a la que se procesan esos datos en las aplicaciones que hacen uso de ellos. Un ejemplo es la nueva barra de búsqueda que incorporó Facebook (Ilustración 2), la cual muestra en tiempo real los resultados que va obteniendo para los términos tecleados en ella. Gracias a tecnologías originadas a raíz de la problemática del Big Data se ha podido abordar este tipo de funcionalidad.

- **Variedad:** Los datos que fluyen por la red son cada vez más variados en tipo y forma. Hay datos en forma de documentos, datos visuales como imágenes o vídeos, datos sonoros… Y datos espaciales, como se tratará en este proyecto. Surge la necesidad de tratar muchísimos datos de tipos muy dispares, por lo que es preciso dotar a los entornos de herramientas para que sean capaces de reconocer esos datos y procesarlos.

Ilustración 2: Barra de búsqueda dinámica de Facebook

URL: http://www.informatica-hoy.com.ar/redes-sociales/imagenes/

Las aplicaciones que se realizan con el fin de cubrir las necesidades que plantea el Big Data surgen para realizar funciones muy específicas. Esto ha propiciado que **no exista una manera única de hacer las cosas**, y que dichas aplicaciones estén en continuo cambio para adaptarse a un entorno cada vez más heterogéneo y en continua trasformación. En función de los objetivos a los que se quiera dar más prioridad han surgido y están surgiendo entornos de una u otra índole, que sean más rápidos procesando *data sets* o que prioricen el análisis de muchos tipos distintos de datos. Estos cambios tan constantes hacen que exista poca documentación práctica sobre cómo iniciarse o manejarse en uno de los numerosos paradigmas de almacenamiento para trabajar sobre Big Data.

A raíz de la necesidad de trabajar sobre gran cantidad de datos de manera mucho más rápida, como en el ejemplo de Facebook, han surgido tecnologías que rompen con los esquemas tradicionales de modelado de datos. Junto con el concepto de Big Data, nace el término No-SQL, con el que se materializa la ruptura con los cánones tradiciones de almacenamiento. Este nuevo concepto engloba todas aquellas tecnologías que no siguen las mismas reglas de

almacenamiento de información que las bases de datos relacionales basadas en SQL. En el próximo apartado se tratarán los conceptos básicos de los dos puntos de vista y sus principales diferencias.

2.2. Bases de datos relacionales y no relacionales. SQL y NoSQL

En la introducción del trabajo se describía que los sistemas de bases de datos relacionales llevan mucho tiempo siendo el modelo informático más empleado del mundo para almacenar y recuperar la información. Desde hace unos años la situación está cambiando. Ahora las necesidades empiezan a ser diferentes, existe mucho más volumen de datos y de tipos muy variados. El conjunto de herramientas que se usan para manejar datos que se han desmarcado de los modelos relacionales que se han diseñado para trabajar teniendo en cuenta los nuevos enfoques sobre los datos han dado lugar al surgimiento del concepto NoSQL y Big Data. ¿Qué diferencias existen entre el modelo relacional basado en SQL y el no relacional?

El modelo relacional ha sido desde principios de los 70 y aun hoy en día continúa siendo la forma habitual de estructurar y almacenar la información en cualquier tipo de aplicación informática, sobre todo en las transaccionales (aquellas aplicaciones con numerosos usuarios que permiten actualizaciones de la BBDD de forma continuada y consultas pocos complejas).
Las bases de datos relacionales cumplen con el modelo relacional, que se fundamenta en el uso de relaciones. Una relación es un vínculo entre dos entidades de la base de datos. La información contenida en las bases de datos relacionales y las relaciones de dependencia que se establecen entre sí, interpretados, persiguen describir el mundo real. El modelo de datos relacional es el modelo actualmente más utilizado para diseñar bases de datos. Se considera una base de datos relacional cuando cumple con el modelo relacional.

El modelo de datos relacional es un modelo matemático de datos. El componente principal del modelo relacional, como ya se ha comentado, son las relaciones. Una relación es una interconexión entre los datos, que a su vez se almacenan en tablas. Así, en sentido inverso, se dirá que dos tablas se relacionan entre sí por la relación que existe entre sus datos. A parte de la relación existe la clave primaria y ajena, los registros, las columnas, y las tablas.

Las bases de datos relacionales cumplen además con las propiedades ACID. Las propiedades ACID (Ilustración 3) garantizan que las transacciones realizadas por el gestor de base de datos para leer o modificar la información se resuelvan de manera predecible y segura, es decir, tal como se prevé que sucedan las operaciones. Esa es la gran ventaja arquitectónica de estos sistemas, aunque también su desventaja principal en términos de coste, al tener que dedicar recursos para garantizar las propiedades. Las cuatro propiedades ACID son las siguientes [2]:

- **Atomicidad** (*Atomicity*): Las operaciones que se lleven a cabo en una transacción (en bases de datos típicamente BEGIN y END) deben de realizarse en todo su conjunto o no realizarse, no pudiendo ejecutarse unas y otras no.

- **Consistencia** (*Consistency*): Integridad de los datos. El estado de los datos debe de ser el mismo antes y después de la transacción. Las restricciones que se apliquen sobre el modelo de datos deben de cumplirse en su totalidad. Por ejemplo, el campo referenciado por una clave ajena debe existir en la tabla a la que se referencia.

- **Aislamiento** (*Isolation*): El sistema que implemente esta propiedad hace que cada transacción parezca ser la única ejecutada, pese a que haya otras ejecutando a la vez. Esto quiere decir que pese a que en realidad se ejecuten concurrentemente varias

transacciones, cada una de ellas no debe ser consciente ni interferir en las fases del resto de ellas.

- **Permanencia** (*Durability*): Esta propiedad garantiza que, si el servicio se suspende en mitad de una transacción y esta queda incompleta, no se pierdan las acciones realizadas tras la reanudación del sistema.

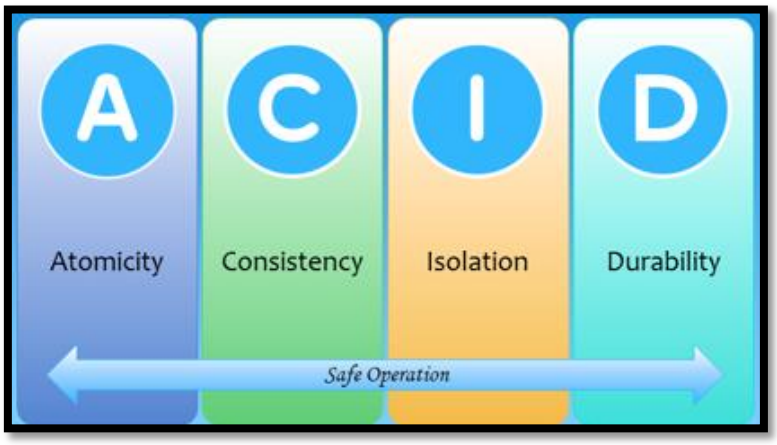

Ilustración 3: Propiedades ACID

URL: http://java.uzmanprogramci.com/acid-transaction-and-deadlock-animated/

En resumen, se puede decir que una base de datos relacional es un conjunto de una o más tablas, estructuradas en registros o tuplas y que se relacionan entre ellas mediante diversas claves. Además, los principios del álgebra relacional y la teoría de conjuntos permiten tratar la información que se quiera almacenar. Por otro lado, las propiedades ACID dotan a estos sistemas de almacenamiento una forma de asegurar la integridad total de las operaciones que se realizan sobre ellas.

En el otro extremo del tablero de juego se ha contemplado que en los últimos años, una gran variedad de bases de datos NoSQL han salido a la luz, creadas por compañías para cubrir necesidades de sus aplicaciones que las bases de datos relacionales no son capaces de dar por sí mismas. Temas como escalabilidad, rendimiento, mantenimiento, etc. que no encontraban en ninguna solución que existía en el mercado. En la web de MongoDB, uno de los principales precursores del NoSQL, se puede encontrar una definición del término muy acertada y fácil de comprender [6]:

"NoSQL encompasses a wide variety of different database technologies that were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs."

La definición de *MongoDB* define el *NoSQL* como el conjunto de bases de datos que destacan por haber sido concebidas en respuesta al gran crecimiento de datos disponibles en los últimos tiempos, la frecuencia con que son accedidos dichos datos y el surgimiento de otras necesidades de procesamiento y tratamiento de esos datos. Puede entenderse como un movimiento rompedor llevado a cabo por dichas empresas para romper con las restricciones impuestas por los sistemas de almacenamiento transaccionales basados en el modelo relacional.

Se puede comprender mejor el movimiento NoSQL con el siguiente ejemplo cotidiano: los tornos de acceso al transporte público, como los establecidos en la red de Metro o Cercanías. Estos tornos sirven para controlar que todos los viajeros que usen el servicio de transporte hayan pagado previamente su billete o estén habilitados mediante abono u otro tipo de tarjeta. Durante todo el día, estos tornos están habilitados y cumpliendo con su función. Existe un personal que se encarga de suministrar servicios a los viajeros, como venta de billetes o atención al cliente, que atienden, entre otras tareas, las incidencias que puedan ocurrir con los tornos de acceso. Esto asegura que el servicio se preste según las normas, que los usuarios dispongan de soporte y que haya menos casos en los que alguien intente acceder al transporte sin pagar. Pero, ¿qué sucede cuando el personal de las estaciones acaba su turno? En muchas estaciones lo normal es que a partir de ese momento no haya nadie de personal, y para no entorpecer el acceso a las vías se dejan abiertos los tornos. Por consiguiente, pueden suceder muchos más casos en los que los usuarios viajen sin pagar, pero por otro lado, se evitan problemas de apertura de tornos ocasionados con la lectura de los billetes/tarjetas, u otros problemas mecánicos. Es decir, **se prioriza el fluido funcionamiento del servicio en detrimento del cumplimiento de las normas**.

Este símil ejemplifica muy bien el surgimiento de NoSQL frente a SQL, el dejar a un lado las transacciones inflexibles y los esquemas de datos fijos, factores que ralentizan las operaciones, y así trabajar directamente sobre el *data set*, obviando reglas ACID (no en todos los casos), sin garantizar plenamente propiedades como la integridad o la consistencia/coherencia, pero ganando en velocidad y simplicidad. En resumen: El movimiento NoSQL es una alternativa que surgió de forma prácticamente clandestina para solucionar problemas concretos, y que en la actualidad es todo un movimiento con comunidades detrás de él que le proporcionan cada vez mayor soporte.

Dicho esto, frente a ACID se plantea otro modelo de transacciones, donde se pierden las propiedades de consistencia y aislamiento a favor de la disponibilidad, la degradación y el rendimiento [2]. Este nuevo modelo se ha denominado BASE. Sus siglas se construyen con las siguientes propiedades:

- ***Basic Availability***: Siguiendo los principios de distribución y replicación se consigue que el sistema siga funcionando pese a que alguna de sus partes no funcione.

- ***Soft State***: No siempre se garantiza la consistencia entre nodos.

- ***Eventual Consistency***: La consistencia de la información se provee de forma eventual, pudiéndose dar o no de manera libre.

Las soluciones NoSQL que se pueden encontrar en el mercado son muy dispares entre sí. Tanto es así que en poco tiempo dichas soluciones se han podido englobar en varios paradigmas de almacenamiento, cada uno aportando un tipo de solución distinta y con sus propias características. Los caminos principales sobre los que avanza el universo NoSQL en la actualidad son los cuatro siguientes:

- **Basadas en clave-valor**: Es el modelo NoSQL más básico. Consiste en almacenar la información en pares clave-valor, donde la clave es un identificador del campo valor al que hace referencia. Si se quiere almacenar más de un valor por clave, puede o bien incluirse los distintos campos separados por campos o bien crear un contenedor por

cada valor distinto, lo cual empeora la eficiencia. Este paradigma prima la disponibilidad de los datos antes que la consistencia. Un ejemplo de aplicación que use este paradigma es Amazon ‘s Dynamo.

- **Basadas en *column-family***: Este paradigma surgió de las *Big Table* de Google. Consiste en que cada clave, a diferencia que en clave-valor, contiene referencias de varias sub claves, llamadas columnas. Esto hace que cada clave tenga distintos números de columnas, por lo que es un modelo de organización flexible. Un ejemplo de este tipo de sistema es HBase.

- **Basadas en documentos**: Paradigma basado en clave-valor. En este caso, el valor consiste en un documento. Un documento es un conjunto de atributos con valores asociados. Además de buscar por la clave, se puede hacer por los atributos del documento, así como indexar por ellos, lo que hace ideal a estos sistemas cuando se necesite buscar no sólo por la clave, sino también por atributos de los documentos. MongoDB es el principal ejemplo de este paradigma, además de ser uno de los sistemas NoSQL más utilizados y vanguardistas.

- **Basadas en grafos**: Basan su estructura en un conjunto de nodos que almacenan pares clave-valor. Dichos nodos están conectados por una serie de aristas, las cuales en este modelo son llamadas relaciones, que indican qué nodos están unidos y por qué. La relación entre dos nodos tiene un nodo del que parte la relación, otro destino y un sentido de la relación, además de un tipo y una descripción de la relación que mantienen entre ellos. Neo4j es la principal referencia en sistemas de almacenamiento basados en grafos.

A modo de resumen, en la Tabla 1 [6] se muestra una comparativa de los principales aspectos definitorios de bases de datos SQL y NoSQL, lo que da una idea mucho más clara de las principales diferencias que existen entre los dos puntos de vista. Esta comparativa se hace analizando los siguientes atributos:

- Tipos: Indica si existen diferentes modelos distintos.

- Historia: Indica la longevidad o no del modelo.

- Ejemplos: Bases de datos de cada tipo de enfoque.

- Modelo de almacenamiento de datos: Forma de almacenar la información, estructuras, datos.

- Esquemas: Aspectos de actualización rente a cambios.

- Escalabilidad: Capacidad de crecimiento de las bases de datos.

- Licencia de software: Modelos de desarrollo que se aplican en uno y otro modelo.

- Transacciones soportadas: Operaciones a nivel de gestor de base de datos que están soportadas.

- Consistencia: Garantía de que las operaciones se realizan en el orden preestablecido.

Tabla 1: Comparativa SQL/NO-SQL

	Bases de datos SQL	Bases de datos NO-SQL
Tipos	Sólo existe un paradigma conocido, el SQL. Las distintas implementaciones del mismo consisten en aplicar pequeñas variaciones al modelo original.	Existen numerosos paradigmas distintos: clave-valor, bases de datos documentales, bases de datos basadas en grafos, *colum family*, etc.
Historia	Surge a principios de la década de los años setenta, con el nacimiento de las primeras aplicaciones dedicadas al almacenamiento de información.	Surgieron a principios de milenio, como medio para paliar las deficiencias del SQL tradicional, tales como la escalabilidad, la replicación o la desestructuración de los datos.
Ejemplos	MySQL, PostGres, Oracle Database, Microsoft Windows Server	MongoDB, Cassandra, HBase, Neo4j
Modelo de almacenamiento de datos	Registros individuales son almacenados en filas en las tablas de base de datos, donde cada columna de la tabla almacena un valor distinto de ese registro. Distintos tipos de relaciones son almacenados en distintas tablas, para luego ser unidas con el objetivo de buscar información que requiera de ambas.	Varios tipos de modelos de datos. Las bases de datos basadas en clave-valor guardan similitud con las SQL, sólo que almacenan la información en sólo dos columnas, una para la clave y otra para el valor de esa clave.
Esquemas	Estático. Cuando se introduce un cambio fijo en un dato de un determinado objeto de la base datos se debe parar el servicio y aplicar el cambio.	Dinámico. Datos que no sean similares entre sí pueden ser actualizados a la vez si es necesario.
Escalabilidad	Vertical. Esto quiere decir que si la base de datos se hace muy grande se aumentará la potencia de cálculo del servidor para agilizar los cálculos. Puede aplicarse escalabilidad horizontal, pero requiere mucho más coste.	Horizontal. Esto quiere decir que si la base de datos crece y precisa de más capacidad, el administrador puede fácilmente configurar los parámetros del sistema.
Licencia de software	*Open source* (MySQL, PostGres) y no *open source* (Oracle Database, Windows Server).	*Open source*.
Transacciones soportadas	Soporta todo tipo de transacciones. Las actualizaciones pueden configurarse como completas o no del todo.	En ocasiones y a ciertos niveles.
Consistencia	Posibilidad de ser configuradas para disponer de una fuerte consistencia.	Depende del producto. Algunos proveen fuerte consistencia (MongoDB) y otros de forma eventual (Cassandra).

Esta comparación de ambos modelos de almacenamiento permite concluir que NoSQL no ha nacido para sustituir a los modelos relacionales basados en SQL. Ambos modelos cubren necesidades distintas, y en función de las características a las que se necesite dar preferencia se elegirá implementar uno u otro.

2.3. Bases de datos espaciales

Tal como definió el profesor Ralf Hartmut Güting de la Universidad a distancia de Hagen (Alemania) [7], las bases de datos espaciales son "*sistemas de almacenamiento que ofrecen el uso de tipos de datos espaciales en el modelado de datos además de un lenguaje de consultas*

capaz de soportarlos. Estos sistemas al menos deben de permitir la indexación espacial y métodos que permitan la unión de tablas por medio de datos/funciones espaciales". Esta definición engloba muy bien lo que un sistema de almacenamiento espacial debe de aportar como extra para manejarse con datos espaciales.

Estos sistemas de almacenamiento surgen de la necesidad de considerar las referencias geográficas y/o geométricas como valor añadido en un conjunto de datos. Algunas bases de datos han sido diseñadas específicamente para el almacenamiento y explotación de datos geográficos, como son SpaceBase o GeoMesa; sin embargo, la mayoría de implementaciones de índole espacial que usan los sistemas más populares de almacenamiento han añadido herramientas en forma de bibliotecas o *plugins* para trabajar con geometrías y geografías. Ejemplos de estas últimas son PostGres o Hive, las cuales incorporan el estándar GIS. Como se verá, PostGres tiene un completo *plugin* para datos espaciales, llamado PostGIS, mientras que Hive adquiere las mismas posibilidades espaciales GIS mediante bibliotecas Java creadas por la comunidad ESRI de Github.

Con el objetivo de facilitar el uso y manejo de datos espaciales y geográficos se suele tomar como base un Sistema de Información Geográfica (GIS). Un GIS proporciona herramientas de análisis y/o visualización de datos geográficos. Los datos geográficos no son más que datos geométricos cuyo sistema de referencia es la superficie de la Tierra, y una de las tareas de estos GIS es traducir geometrías en geografías. En este trabajo práctico se ha utilizado en ocasiones la aplicación de escritorio QGIS, con la cual se pueden visualizar distintos tipos de geometrías, como los puntos (Ver *Anexo IV* para más información sobre QGIS).

En los puntos que se describen a continuación se detallarán aspectos específicos de PostGres y Hive. Pese a que este trabajo práctico está centrado en Hive, es conveniente profundizar en algunos puntos del estándar PostGIS, ya que en gran medida las funciones de las bibliotecas espaciales desarrolladas para Hive por ESRI contenidas en GitHub están basadas en las que existen en PostGIS, por lo que funcionan de manera idéntica en muchos casos.

2.3.1. PostGIS y GIS Tools Hadoop

PostGIS es una extensión al sistema de base de datos objeto-relacional PostgreSQL. Permite el uso de objetos GIS (*Geographic Information Systems*). PostGIS incluye soporte para índices GiST basados en R-Tree, y funciones básicas para el análisis de objetos GIS. Esta creado por Refractions Research Inc., como un proyecto de investigación de tecnologías de bases de datos espaciales. Está publicado bajo licencia GNU [3]. En el *Anexo I* se puede consultar la forma de instalar y configurar esta extensión en PostGres.

Por otro lado, GIS Tools for Hadoop consiste en un conjunto de librerías de código abierto orientadas al análisis de Big Data. Estas son desarrolladas, mantenidas y actualizadas por ESRI, compañía que posee el treinta por ciento del mercado de software GIS. Entre esas librerías se cuenta con las dos siguientes, las cuales han sido la base para el trabajo de este proyecto:

- *Esri Geometry API for Java*: Librería geométrica espacial estándar, que puede ser usada para añadir a Hadoop la capacidad de tratar tipos de datos y operaciones de naturaleza

geométrica, y permitir a los desarrolladores construir aplicaciones basadas en *MapReduce*. Se corresponde con la capa de funciones de GIS Tools a más bajo nivel. Como se detallará en el caso práctico, ésta librería Java corresponde con la *esri-geometry-api.jar*, usada en el caso práctico. No será objeto de estudio en este trabajo.

- *Spatial Framework for Hadoop*: Librería que habilita a Hive y a los usuarios de Hive Query Language (HQL) interactuar con datos y funciones espaciales varias. Esta librería se corresponde con la *spatial-sdk-hive.1.0.1.jar*, utilizada en este trabajo. Es la capa de funciones más cercana al usuario de Hive. Varias de las funciones que contiene esta librería serán objeto de estudio en el punto *2.3.3. Funciones espaciales en PostGIS y Hive*.

La Ilustración 4 muestra los dos niveles de abstracción en los que se desenvuelven las dos bibliotecas espaciales en Hadoop/Hive. La Spatial Framework contiene las funciones que el usuario utilizará en HQL cuando realice consultas espaciales, y la Esri Geometry API se encarga de realizar dichas consultas de la manera más eficiente, según el tipo de datos que se manejen y las funciones y tipos de cláusulas que se utilicen en las consultas. En definitiva, la primera es el qué se hace y la segunda es el cómo se hace.

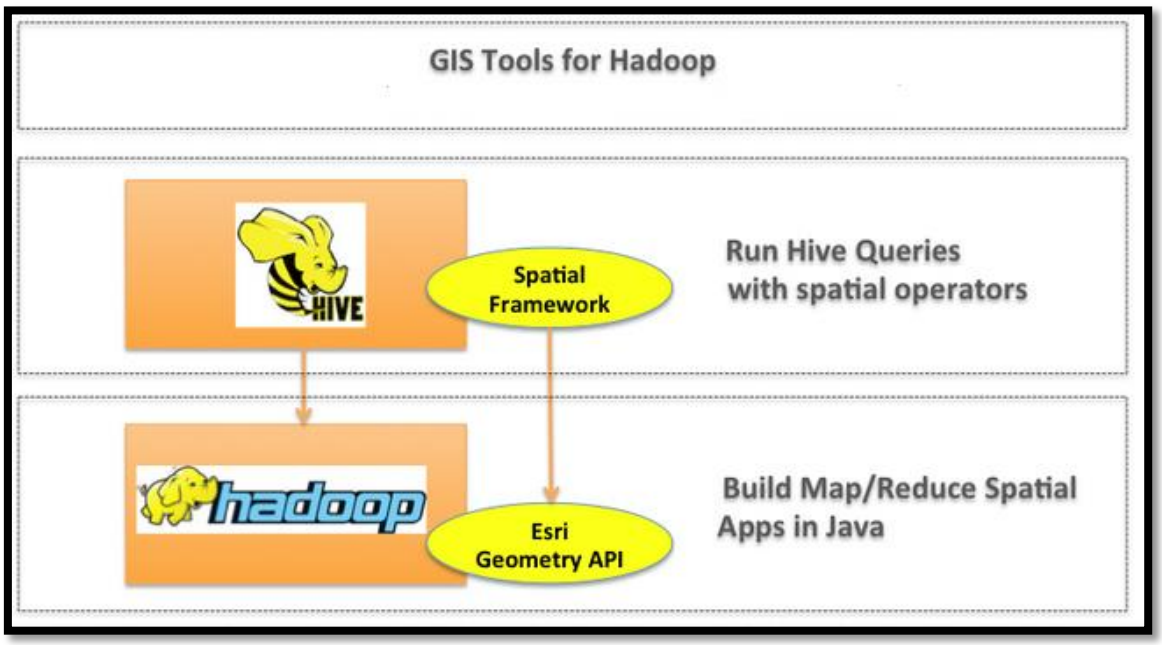

Ilustración 4: Niveles de abstracción en GIS Tools

URL: http://esri.github.io/gis-tools-for-hadoop/

2.3.2. Datos espaciales en PostGIS y Hive

El estándar OpenGIS define el tipo de datos que se pueden manejar en una base de datos espacial. El Estándar es similar para PostGIS y para las bibliotecas espaciales de ESRI en Git Hub para Hive, existiendo diferencias mínimas entre ellos.

Cuando en terminología espacial se habla de datos espaciales, en el contexto de los sistemas que manejan datos espaciales en realidad se está haciendo referencia a objetos espaciales. Un objeto espacial es la representación geométrica de una localización real dada en forma de figura geométrica. El estándar OpenGIS define qué objetos espaciales se pueden construir en un sistema espacial. En la Tabla 2 [8] se pueden ver los objetos espaciales más comunes que establece el estándar OpenGIS así como ejemplos de los mismos.

Tabla 2: Objetos espaciales OpenGIS

Objeto espacial	Descripción	Ejemplo
POINT	Punto simple	POINT (0,0)
LINESTRING	Línea de puntos	LINESTRING (0 0, 1 1, 1 2)
POLYGON	Polígono de puntos	POLYGON (0 0 0,4 0 0,4 4 0,0 4 0,0 0 0)
MULTIPOINT	Conjunto heterogéneo de puntos	MULTIPOINT (0 0 0,1 2 1)
MULTULINESTRING	Conjunto heterogéneo de líneas	MULTILINESTRING ((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
MULTIPOLYGON	Conjunto heterogéneo de polígonos	MULTIPOLYGON (((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0), (1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)), ((- 1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
GEOMETRYCOLLECTION	Conjunto heterogéneo de cualquier objeto espacial	GEOMETRYCOLLECTION(POINT(2 3 9), LINESTRING((2 3 4,3 4 5))

La representación de los ejemplos anteriores está expresada en *Well Known Text* (WKT). OpenGIS define dos formas de representar los objetos espaciales:

1. *(WKT)Well Known Text*: Forma canónica para expresar geometrías con caracteres ASCII, entendible para las personas. Es la forma que se ha utilizado en este proyecto para representar y manejar datos geográficos.

2. *(WKB)Well Known Binary*: representación portable de un valor geométrico, representado por una ristra de bytes. Esta forma es utilizada para intercambiar valores geométricos en binario entre una base de datos SQL y un cliente Oracle

Las dos formas guardan información del tipo de objeto y sus coordenadas. Además la especificación OpenGIS requiere que los objetos incluyan el identificador del sistema de referencia espacial (SRID). El SRID es requerido cuando se inserta un objeto espacial en la base de datos, y también lo usan algunas funciones espaciales como metadato. Como se verá, no siempre es necesario incluirlo, aunque sí es recomendable.

2.3.3. Funciones espaciales en PostGIS y Hive

Las funciones espaciales aúnan un conjunto de funcionalidades dedicadas a la explotación de datos de índole espacial. La Tabla 3 contiene un resumen de las funciones espaciales más importantes utilizadas en las bases de datos espaciales. Para introducirlas se ha decidido incluir la siguiente información al respecto:

- Función espacial: Nombre de la función y parámetros que recibe. Nótese que todas las funciones están nominadas con el prefijo "ST", el cual indica la propiedad de Standard.
- Descripción: Breve introducción a la funcionalidad de la función, los datos que devuelve y los datos de entrada a la función.
- Hive: Indica si existe una implementación parecida o igual de las funciones existentes para PostGres en Hive o no.
- Usada en caso práctico: Indica si la función en cuestión se ha utilizado en el caso práctico o no.

Tabla 3: Principales funciones espaciales

Función espacial	Descripción	Hive	Usada en caso práctico
ST_Area (geom)	Devuelve el área de un polígono o multipolígono en unidades de SRID (geometrías) o metros cuadrados (geografías).	✓	✓
ST_AsText (geom)	Devuelve la representación en *Well Known Text* de una geografía/geometría dada.	✓	✓
ST_Contains (geomA, geomB)	Devuelve verdadero si al menos un punto de B reside dentro de A o si todo B está en A.	✓	✓
ST_Crosses (geomA, geomB)	Devuelve verdadero si algún punto o región de B reside en A, pero no todos ellos.	✓	✗
ST_Disjoint (geomA, geomB)	Devuelve verdadero si las dos regiones no intersectan, es decir, si no comparten ninguna región.	✓	✗
ST_Distance (geomA, geomB)	Devuelve la mínima distancia existente en grados cartesianos entre dos geometrías/geografías.	✓	✓
ST_Dwithin (geomA, geomB, distance)	Devuelve verdadero si las dos geometrías/geografías están a igual o menor distancia que la indicada en el parámetro distancia. El parámetro se indica en unidades geométricas si se comparan geometrías y en metros si se comparan geografías.	✓	✗
ST_Equals (geomA, geomB)	Devuelve verdadero si ambas geometrías son iguales.	✓	✓
ST_GeometryType(geom)	Devuelve el tipo de la geometría pasada por parámetro.	✓	✓
ST_GeomFromText (WKT geom)	Devuelve el valor de la geometría de una determinada expresión *Well Known Text*.	✓	✓
ST_Intersects (geomA, geomB)	Devuelve verdadero si dos formas geométricas 2D coinciden en algún punto en el espacio.	✓	✓
ST_Length (geom)	Devuelve la longitud en grados cartesianos de un *LineString* o *MultiLineString*.	✓	✓
ST_Linestring (WKT geom)	Devuelve el valor de un *LineString* a partir de su	✓	✗

ST_Linestring (geom points)	forma Well Known Text, o a partir de sus puntos geométricos.		
ST_Overlaps (geomA, geomB)	Devuelve verdadero si ambas geometrías coinciden en tipo de dimensión y si tienen en común alguna zona o punto.	✓	✗
ST_Point (cordX, cordY)	Devuelve el valor geométrico de un punto a partir de sus coordenadas.	✓	✓
ST_Polygon (WKT Linestring geom) **ST_Polygon (points geom)**	Devuelve el valor geométrico del polígono dado a partir o bien de un *LineString* en *Well Known Text* o bien a partir de sus coordenadas geométricas.	✓	✗
ST_SetSRID (geom, SRID)	Añade a los metadatos de la geometría indicada en su primer parámetro el SRID indicado mediante su segundo parámetro.	✓	✓
ST_Transform (geom, SRID)	Devuelve una nueva geometría a partir de una dada, cuyas coordenadas estarán dadas en el sistema de referencia indicado (SRID).	✗	✗
ST_Touches (geomA, geomB)	Devuelve verdadero si las geometrías tienen en común al menos un punto, pero no intersectan por dentro de las geometrías.	✓	✗
ST_Within (geomA, geomB)	Devuelve verdadero si B está totalmente dentro de A.	✓	✗

Las funciones espaciales utilizadas en el caso práctico serán descritas con mayor profundidad en su ámbito de uso cuando se explique su utilización en las consultas espaciales en las que intervengan (ver 3.*6. Generación asistida de consultas espaciales*).

2.4. Hadoop/Hive y Hive Query Language

Hive es una interfaz software de licencia libre construido sobre Apache Hadoop, la cual está destinada a trabajar sobre arquitectura Map Reduce. Su función principal es facilitar la consulta y la administración de datos masivos en un medio de almacenamiento distribuido, como es *Hadoop File System* (HDFS). Hadoop provee alta escalabilidad y tolerancia a fallos para el almacenamiento y el procesamiento de datos gracias al uso del paradigma *Map Reduce*. Con dicho fin, Hive cuenta con su propio lenguaje, el cual imita a SQL, llamado HiveQL (o HQL).

Hadoop fue concebido para organizar y almacenar información de forma masiva de todos los tipos, tamaños y formatos. La orientación de su arquitectura hacia la lectura de información hace que pueda usarse como un perfecto almacén de datos heterogéneos, tanto estructurados como no estructurados. Y aquí es donde entra Hive, el cual proporciona a los analistas de datos una manera sencilla y eficaz de interactuar con Hadoop mediante su ya mencionado lenguaje de consulta HQL. Hive proporciona sobre Hadoop:

- Familiaridad HQL-SQL: Al concebirse como un símil del estándar SQL, HiveQL proporciona una interfaz amigable para los desarrolladores que provengan de entornos relacionales.

- Rapidez: Incluso en *data sets* que contengan volúmenes de información masivos, Hive garantiza tiempos de respuesta interactivos, garantizando una buena usabilidad.

- Escalabilidad y extensibilidad: Un aumento de la variedad y cantidad de datos almacenados en Hadoop no merman la eficiencia del sistema, es más, Hadoop Hive es más óptimo cuanto mayor sea la carga de trabajo. Esto se debe a que en esencia, Hadoop utiliza *Map Reduce*, y este realiza procesamiento en paralelo y distribuido de los datos. A mayor carga de trabajo, mayor distribución del mismo.

Hive de cara al usuario trabaja igual que cualquier sistema relacional de almacenamiento de datos. Los datos se organizan en tablas, y las tablas están construidas sobre particiones. Los datos de las tablas pueden ser accedidos vía consultas. Con respecto a Hadoop, cada tabla creada con Hive se corresponde con un directorio de HDFS. Cada tabla puede dividirse en particiones que determinan cómo se distribuyen los datos en los sub directorios dentro del directorio principal.

Otras capacidades que posee Hive son:

- Herramientas para la fácil extracción/carga/lectura de datos. Ejemplo: Apache Pig.

- Acceso a archivos almacenados directamente en HDFS u otros repositorios de datos como HBase.

- Ejecución de consultas usando *Map Reduce*.

- Uso de funciones personalizables (UDF), agregaciones (UDAF's) y funciones de tabla (UDTF's). En el caso práctico será necesario crear funciones personalizables para utilizar las funciones espaciales a partir de las bibliotecas espaciales.

- En su última versión, 0.14, incorpora la posibilidad de añadir propiedades ACID a las transacciones.

En definitiva, Hive es una capa intermedia entre usuario y HDFS, la cual facilita el acceso y manipulación de los datos sin necesidad de comprender los entresijos de Hadoop. Gracias a su lenguaje HQL construido para imitar el SQL relacional, aquellos usuarios con nociones sobre SQL pueden de forma intuitiva trabajar sobre Hadoop a través de Hive. En los siguientes puntos se describirán los aspectos fundamentales de este lenguaje para facilitar la comprensión del caso práctico: tipos de datos y principales comandos de creación y manipulación de datos (DDL y DML).

2.4.1. Tipos de datos en HQL

Hive soporta muchos tipos primitivos de datos conocidos en otros lenguajes de implementación o de consulta, tales como los derivados del *Integer* o el *Boolean*, pasando por los *Char* o *Binary*. No obstante, también permite la combinación de tipos primitivos para crear formas de datos más

complejas como las estructuras, los *arrays* y los mapas. En este punto se pondrá el foco de atención en los tipos primitivos, los cuales han sido utilizados en el caso práctico.

La Tabla 4 [9] recoge los tipos de datos básicos soportados en Hive. Se da una descripción del tipo de dato. Todos ellos pueden usarse como tipo de dato en las columnas de las tablas o en los comandos de manipulación o definición del lenguaje.

Tabla 4: Tipos de datos básicos en HQL

Tipo de dato	Descripción
TYNINT	Entero con signo de 1 byte de tamaño.
SMALLINT	Entero con signo de 2 bytes de tamaño.
INT	Entero con signo de 4 bytes de tamaño. Por defecto se toma INT para enteros, excepto para cuando se excede el tamaño límite, en cuyo caso se interpreta como BIGINT.
BIGINT	Entero con signo de 8 bytes de tamaño.
FLOAT	Coma flotante de precisión simple de 4 bytes de tamaño.
DOUBLE	Coma flotante de precisión doble de 8 bytes de tamaño.
DECIMAL	Coma flotante de precisión personalizable por el usuario.
TIMESTAMP	Unix Timestamp con precisión opcional de nanosegundos. Admite conversiones de enteros, decimales y *strings.*
DATE	Describe una fecha particular dada en año-mes-día (YYYY-MM-DD).
STRING	Cadena de longitud variable de texto. Pueden acotarse mediante el uso de comillas simples (‘) o dobles (‘’). Los espacios en blanco son tenidos en cuenta en las comparaciones entre cadenas.
VARCHAR(MaxLength)	Cadena de longitud variable de texto creada con un límite de tamaño (de 1 a 65355 caracteres). Los caracteres en blanco son tenidos en cuenta en la longitud y comparaciones entre cadenas.
CHAR(Length)	Cadena de longitud fija creada con un tamaño invariable. Si la cadena almacenada en una columna de este tipo no llega al mínimo establecido, se realiza *padding* con espacios en blancos hasta completar la longitud.
BOOLEAN	Tipo con dos valores a tomar, *true* o *false*.
BINARY	Valor binario.

Hive asigna valor NULL a aquellos campos sin ningún valor. Por último, admite *castings* entre tipos de datos tanto forzados como automáticos. Para forzar conversiones cada tipo tiene su propio comando, el cual consiste en el nombre del tipo seguido de ‘*to*’ (Ejm: *int to*). Puede consultarse la tabla completa de conversiones entre tipos proporcionada en el manual oficial de de Hive [9].

2.4.2. Lenguaje de definición y manipulación de datos (DDL y DML) de HQL

Al igual que en un SQL relacional, en Hive se puede hablar de dos tipos de operaciones básicas: por un lado, aquellas que permiten la definición de estructuras de datos; y por otro lado, aquellas que permiten manipular dichas estructuras.

En HiveQL existen, al igual que en otros lenguajes de base de datos, comandos para la creación, modificación y eliminación de tablas y esquemas de datos. Las operaciones básicas en ambos casos son la terna *Create/Alter/Drop*. Estas operaciones pueden ir acompañadas de atributos extra para complementar su funcionamiento. Por ejemplo, puede indicarse con EXTERNAL al crear una tabla que no se almacenará en el directorio por defecto de HDFS.

Por otro lado, en Hive existen todo tipo de mandatos para la manipulación de los datos. Existen cuatro que permiten editar el contenido de las tablas, los cuales son: *Insert/Update/Delete/Load*. Por otro lado, los comandos de consulta permiten extraer datos de las tablas de múltiples maneras. En Hive se parte de la cláusula SELECT/FROM para hacer una consulta de atributos sobre una tabla, y a partir de ella pueden utilizarse otras para filtrar los datos, entre las que se encuentran GROUP BY, HAVING o WHERE. Hive también permite el uso de sub consultas mediante el uso de alias para agruparlas, además de poderse realizar combinaciones entre tablas mediante la cláusula JOIN.

En esencia, HQL tanto en DDL como DML son muy parejos a cualquier SQL de un sistema Gestor de Bases de Datos relacional. Para consultar más información acerca de ambos aspectos consultar el apartado dedicado a tal efecto en el manual oficial de Hive [10].

3. Descripción del caso práctico

A lo largo de este apartado se describe todo el trabajo realizado. Está dividido en distintos puntos que agrupan distintas etapas de la parte práctica del proyecto. Los puntos son:

- Instalación y configuración de entornos: Se detalla cómo instalar y dónde encontrar los recursos con los que se ha trabajado, así como guiar en la configuración de paquetes, variables de entorno y otros aspectos técnicos.

- Explicación general del caso práctico: Se da una imagen global de lo que ha sido el proyecto, sin profundizar en detalles. Se habla sobre los patrones de los que se parte para construir consultas.

- Explicación específica del caso práctico: Se describe de dónde se parte, el porqué de las decisiones del caso práctico y los aspectos más importantes sobre los datos de partida.

- Creación de base de datos: Se explica las decisiones de diseño de tablas y cómo se generan en Hive.

- Importación de los datos: Detalles acerca de las decisiones a la hora de cargar los datos en las tablas de Hive a partir de los archivos CSV.

- Generación asistida de consultas: El punto central del trabajo. Se habla del diseño del esquema de generación de consultas, sus ramas, consultas de ejemplo detalladas y explicación de cada resultado.

- Pruebas de consultas: Se incluyen algunas pruebas sobre determinadas consultas como forma de corroborar que éstas funcionan correctamente.

3.1. Instalación y puesta a punto del entorno de trabajo

El trabajo realizado en este proyecto ha necesitado de la instalación y configuración de distintos entornos software. Parte del trabajo del caso práctico se ha dedicado a desplegar estos entornos. Por esta razón se incluyen los procesos de instalación y configuración básica de estos entornos. También se incluye una breve introducción sobre cada herramienta. Con el objeto de no entorpecer la lectura de este documento, estos procesos se han incluido en los siguientes anexos:

- **Instalación de PostGres con soporte a datos espaciales (Anexo I)**
- **Instalación de Hive en Windows con Sandbox de Hortonworks (Anexo II)**
- **Instalación de Hadoop(Hive/HBase) en Windows Server (Anexo III)**
- **Instalación de QGIS (Anexo IV)**

Con la ayuda de estos anexos, el lector, si así lo desea, dispondrá de ayuda para localizar los recursos y resolver las cuestiones más críticas de la instalación y configuración de todos los paquetes de software utilizados para la elaboración del caso práctico.

3.2. Explicación general del caso práctico

El núcleo del proyecto que se documenta en esta memoria consiste en el seguimiento de patrones únicos ya establecidos con los que, a partir de ellos, poder diseñar consultas espaciales para Hive que resuelvan problemas de búsqueda de datos sobre un *data set* que contenga atributos geográficos. Los patrones ya son conocidos, puesto que se toman de otro trabajo fin de grado realizado por otro estudiante de la Universidad. Considerando esto, **el trabajo se centra en justificar que esos patrones de los que se parten para diseñar consultas espaciales son únicos e irrevocables,** no existiendo ninguna otra forma para elaborar soluciones más allá de las obtenidas a partir de dichos patrones. El medio para conseguirlo es utilizar un **esquema de implementación de consultas**, obtenido a partir de la experimentación múltiple realizada sobre Hive.

Los patrones de los que se parte se han obtenido del Trabajo Fin de Grado de Sergio Casillas Cortázar, *Transformación de las restricciones OCL de un esquema UML a consultas de SQL*. En dicho TFG en parte se hace un trabajo parecido al de este proyecto, ya que se desgranaba el lenguaje OCL en sus distintas posibilidades. Para esa clasificación OCL se usaban tres patrones básicos, patrones de los que se parte en este proyecto para empezar a experimentar, los cuales son los siguientes:

- **Consultas simples (Una tabla):** Consultas que trabajan sobre una única tabla, seleccionando datos de forma básica, usando funciones o usando cláusulas para filtrar los resultados.

- **Combinaciones sobre misma tabla:** En este caso se practica la combinación de distintas versiones de una misma tabla, para enfrentar varios subconjuntos de datos de la misma.

- **Combinaciones sobre tablas distintas:** Consultas que aúnan combinaciones entre distintas tablas.

Para justificar que esos patrones son únicos y válidos por sí mismos, se ha diseñado un caso práctico, cuyo fin es obtener una batería de consultas clasificadas según ciertos criterios (tabla de datos a la que afecta, resultados, etc.). Esta batería de consultas será lo suficientemente amplia y variada como para cubrir sobradamente las posibilidades de los tres patrones de diseño para las tablas que se han diseñado específicamente en este trabajo.

Las consultas serán lanzadas sobre tres tablas de Hive, relacionadas entre ellas exclusivamente por los atributos geográficos. Estas tres tablas han recibido una serie de modificaciones sobre su estructura y datos, las cuales se explicarán a partir del siguiente apartado. Todo el proceso de creación de los entornos de trabajo, creación de las tablas e importación de datos se explican al detalle en las páginas siguientes.

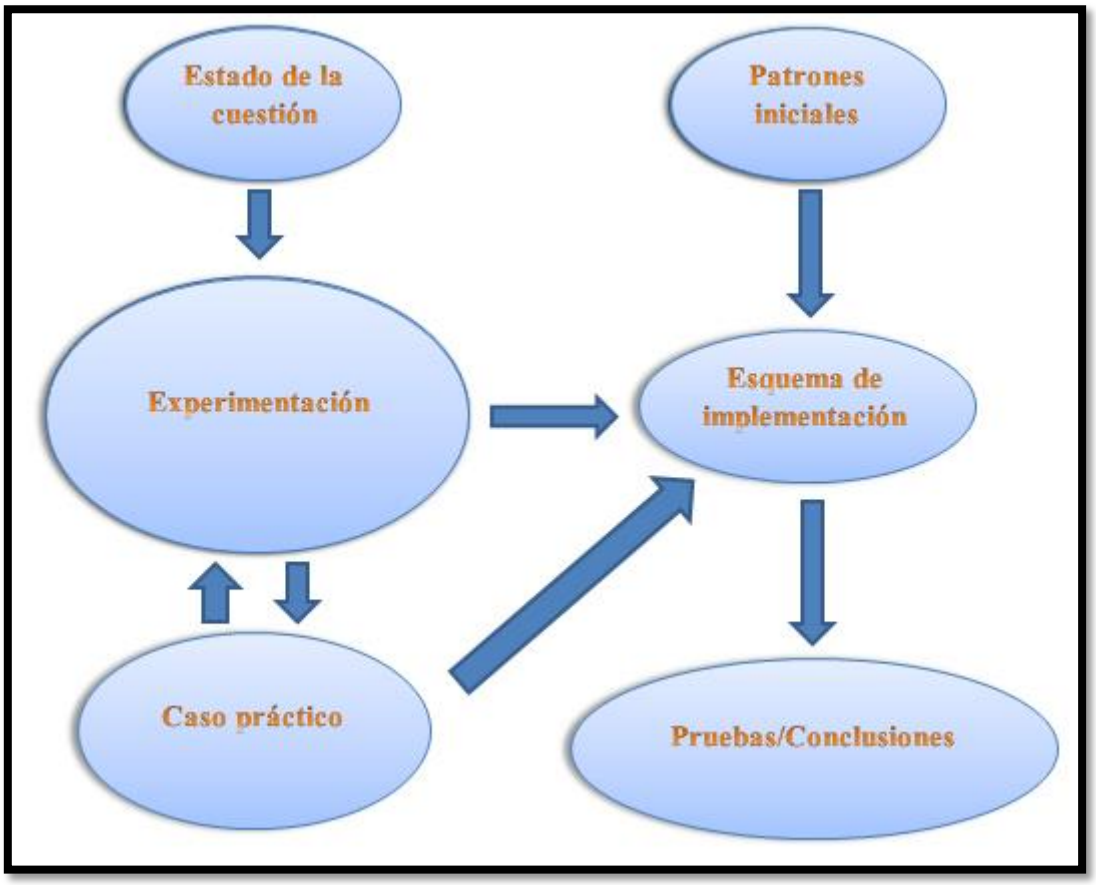

Ilustración 5: Pasos del trabajo y flujos entre tareas

En la Ilustración 5 puede verse las distintas tareas de las que consta el proyecto y el flujo entre ellas. Inicialmente se experimenta en Hadoop/Hive conociendo las bases teóricas únicamente. Una vez que se ha experimentado y se conoce de forma básica como funciona la herramienta, se diseña un caso práctico sobre el que trabajar de forma más ordenada y así centrar el problema en una casuística concreta. Se vuelve a experimentar ya sobre el caso práctico de forma masiva, y de esta manera se extrae un conjunto de información que nos lleva a diseñar el esquema de implementación. El último paso es realizar pruebas y analizar los resultados del estudio.

3.3. Explicación específica del caso práctico

Con el objetivo de marcar una base sobre la que operar mediante la implementación y ejecución de consultas espaciales en Hive, se ha diseñado un caso práctico. Dicho caso práctico consta fundamentalmente de tres tablas de datos relacionados entre sí única y exclusivamente por sus campos de datos espaciales. De entre estas tres tablas la más importante y sobre la que gira todo el trabajo es la que contiene los datos de geolocalización de la aplicación Uber. Uber es una empresa internacional propietaria de la aplicación para móviles con el mismo nombre Uber. Dicha aplicación tiene como usuarios a conductores y viajeros, los cuales se comunican a través de la aplicación para planificar viajes de distinta índole y distancia.

El fichero con los datos masivos Uber del que se parte es *Uber.csv* (*7. Bibliografía, Otras fuentes de interés*). Dicho fichero se ha obtenido de InfoChimps. Dicho fichero contiene registros en forma de coordenadas, hora y año de la toma de los mismos, así como el identidicador de ruta, correspondiente cada uno a un usuario distinto de Uber. El fichero contiene más de un millón de estos registros, suficiente volumen de datos como para ser tratado dentro de un paradigma Big Data. En la Ilustración 6 puede verse un extracto del fichero *Uber.csv*.

26	1,2007-01-07 10:56:50+00:00,37.786564,-122.440209
27	1,2007-01-07 10:56:54+00:00,37.786905,-122.44027
28	1,2007-01-07 10:56:58+00:00,37.786956,-122.440279
29	2,2007-01-06 06:22:35+00:00,37.800224,-122.43352
30	2,2007-01-06 06:22:39+00:00,37.800155,-122.434101
31	2,2007-01-06 06:22:43+00:00,37.80016,-122.43443

Ilustración 6: Extracto de Uber.csv

Otra de las tablas del caso práctico es *Districts*. Los datos de esta tabla han sido obtenidos modificando el fichero *districts.xlsx* (*7. Bibliografía, Otras fuentes de interés*). Dicho fichero contiene datos de localización de la mayoría de distritos de San Francisco. Los datos de los distritos vienen dados en latitudes norte y sur y longitudes este y oeste. En total se proporcionan cuatro puntos combinando estas medidas, lo que supone es que lo que se tenga sea una aproximación rectangular 2D del distrito.

Por ultimo, la tabla *PointsInterest* es la única no obtenida por fuente externa. Dicha tabla almacena la localización y el tipo de punto de interés de zonas emblemáticas o de importancia de San Francisco, desde museos a hoteles de referencia. Los datos se han obtenido localizando manualmente puntos de interés mediante el uso de Google Maps. Con ellos se contruyó un fichero *PointsInterest.csv* disponible para cargarse en Hive.

El sistema de referencia geográfico utilizado para mostrar las magnitudes geográficas ha sido el WGS84. Este sistema utiliza datos de latitud y longitud para describir localizaciones, pudiendo tener valores para la latitud de entre -90 y 90, y para las longitudes de entre -180 y 80. Su identificador es el 4326.

Antes de empezar a trabajar con cualquier tipo de función espacial es necesario añadir el *jar* que contiene la implementación de dichas funciones, así como crear una función Hive por cada una de las funciones espaciales que se añadan, en ese orden. En el *Anexo V* pueden verse todas las sentencias de creación de funciones Hive que se han utilizado para todas las funciones espaciales manejadas en el proyecto y la adición de las bibliotecas (*jars*).

Se ha utilizado Postgres paralelamente a Hive para distintas cuestiones: exportación de datos, prueba de funciones espaciales y diseño e implementación de consultas, para luego traducirlas a HQL. Esto se ha debido a la mayor rapidez de depuración y ejecución en PostGres, ya que al trabajar sobre un único nodo en Hadoop/Hive los trabajos *Map Reduce* son lentos.

3.4. Creación de base de datos

En este caso práctico se ha trabajado con un total de tres tablas en Hive. Análogamente, estas tablas han sido también creadas en PostGres con el objetivo de trabajar en este sistema de almacenamiento para aligerar algunos trabajos, como por ejemplo la depuración de consultas. También se han cargado los datos en PostGres.

Las tablas creadas son: *Uber, PointsInterest y Districts* Cada tabla tiene adheridas una serie de decisiones de diseño dadas por la naturaleza de los datos de origen y los tipos de datos que soporta Hive. Cabe decir que se han obviado restricciones de integridad referencial y relaciones entre tablas, **puesto que toda relación que exista entre las tres tablas se dará única y exclusivamente mediante su campo/campos geográficos.** Ningún otro campo unirá las tablas en las consultas espaciales. Esto acota la definición de las mismas a indicar su uso, qué campos posee cada una y qué tipo de dato soporta. La tabla 5 refleja las características de estas tablas.

Tabla 5: Descripción de tablas del modelo

Tabla	Descripción	Campos
Uber	Tabla que almacena más de un millón de registros pertenecientes a puntos de localización Uber.	*IdU* → Identificador de una ruta. Tipo INT. *DatetimeU* → Fecha y hora de obtención del punto de localización. Tipo STRING *LatitudeU* → Latitud del punto de la ruta. Tipo DOUBLE. *LongitudeU* → Longitud del punto de la ruta. Tipo DOUBLE.
PointsInterest	Tabla que almacena un conjunto de puntos de referencia de San Francisco: hoteles, locales de ocio, edificios históricos, etc.	*NamePI* →Nombre del punto de interés. Tipo STRING. *TypePI* → Tipo de localización. Tipo STRING. *LatitudePI* → Latitud de la localización.

		Tipo DOUBLE. *LongitudePI* → Longitud de la localización. Tipo DOUBLE.
Districts	Tabla cuyo contenido son los veinte distritos de San francisco con los que se trabaja en el caso práctico.	*NameD* →Nombre del distrito. Tipo STRING. *GeometryD* → Polígono en formato WKT del distrito. Tipo STRING

Se tomaron algunas decisiones de diseño sobre columnas y tipos de datos. Estas decisiones son:

- Sobre *Uber*: Las columnas son las mismas que los datos que aparecían en el fichero *Uber.csv*. El *IdU* es tipo entero, siempre toma valores naturales. *DatetimeU* es tipo *string*, porque para el tipo de consultas que se van a manejar no se precisa un tipo Date. Las latitudes y longitudes son tipo *double* para disponer de mayor precisión respecto a los decimales en los puntos geográficos que si por ejemplo se usase tipo *float*.

- Sobre *PointsInterest*: Esta tabla se diseñó y creó desde cero. Sus campos tipo texto, *namePI* y *typePI* sirven para tener datos cualitativos de cada punto. Ambos campos son de tipo *string*. En cuanto a las latitudes y longitudes se aplica el mismo razonamiento que en la tabla *Uber*, son tipo *double* para reflejar todos los decimales posibles y aumentar la precisión de las coordenadas de los puntos.

- Sobre *Districts:* Los campos que describían cada distrito en un principio eran cinco, siendo estos el nombre y las latitudes norte/sur y este/oeste. Se decidió que para importar los datos de manera uniforme y de la misma manera que en *Uber* y *PointsInterest (3.5. Importación de los datos)*, unido además al hecho de tener que adaptar la geometría para poder usar la función espacial *GeomFromText (2.3.3. Funciones espaciales en PostGIS y Hive)*, se diseñara una tabla con dos columnas, una con el atributo *nameD* con el nombre del distrito y otra con la geometría en formato Well Known Text (WKT).

Crear una tabla en Hive es muy similar a crear una tabla en cualquier lenguaje basado en SQL. La instrucción que se usa en HQL es CREATE, la misma que en SQL estándar. A continuación se muestra un ejemplo de cómo crear la tabla *Uber* en *Hive* según las especificaciones que se han explicado en este punto. Las sentencias completas para todas las tablas pueden verse en el *Anexo V*.

```
CREATE TABLE Uber (idU INT,
                   datetimeU STRING,
                   latitudeU DOUBLE,
                   longitudeU DOUBLE);
```

Además de las tablas principales también se deben crear las tablas intermedias que se usan en la importación (*3.5. Importación de datos)*. Estas tablas se usan para almacenar de forma temporal los datos provenientes de los archivos CSV antes de extraer de ellas las distintas filas para insertarlas en las tablas finales. Estas tablas temporales son muy básicas, y sólo contienen una columna de tipo texto (*string*) que almacena toda una fila de datos. Existe una tabla temporal por cada tabla principal. Un ejemplo es la tabla temporal de *Uber*, de la cual se muestra a continuación su sentencia de creación. El resto de tablas temporales pueden consultarse en el *Anexo V*.

CREATE TABLE uber_temp (col_temp STRING);

Una vez que han sido creadas las tablas con las que se va a trabajar se puede iniciar la carga de datos.

3.5. Importación de los datos

Una vez que las tablas han sido creadas en Hive, se deben cargar sobre ellas los datos sobre los que se trabaja. El proceso de carga final de datos en este caso práctico es resultado de múltiples ajustes realizados sobre la estructura y contenido de dichos datos, el diseño de tablas y los tipos de datos manejados. Esto hace que dicha carga esté enfocada exclusivamente al trabajo realizado en este proyecto.

La carga de datos, pese a parecer un proceso trivial, no lo es. El proceso depende de varios factores a tener en cuenta, entre los que se encuentran: eficiencia, consistencia /coherencia, volumen de datos y naturaleza de los datos. Como este caso práctico está enfocado al uso de HQL como medio de programación de consultas espaciales y su fin no está ligado a optimizar procesos, se ha prestado especial atención a dos de esto factores:

- Volumen de datos: Manejar un millón de datos (tabla *Uber*) no es lo mismo que trabajar sobre una veintena (tabla *Districts*). Pese a que no es objetivo de este trabajo, sí es importante considerar este factor.

- Naturaleza de los datos: Se deben considerar la fuente de los datos (Ejm: Fichero Excel) o cuáles son los tipos de datos soportados en el entorno de trabajo. Este punto es el que marca de forma vehemente el camino a seguir para importar.

La manera en que estos dos factores se presentan en las tres tablas sobre las que se trabaja hace que hayan existido diferencias notables para realizar la carga. El proceso de importación conlleva seguir dos pasos diferenciados hasta tener en Hive los datos disponibles:

1. Obtención de los datos de carga: Para todas las tablas se diseña la estructura de un fichero CSV, el cual incluye los datos de cada una de ellas. Este paso es distinto para cada tabla. Tener los ficheros en este formato facilita la importación de datos mediante el uso de expresiones regulares sencillas.

2. Carga de los datos en Hadoop/Hive: Los ficheros de datos obtenidos se cargan en el sistema de ficheros de Hadoop, para posteriormente copiarlos en el espacio de trabajo de Hive e importar los datos en las tablas usando expresiones regulares para mapear los valores de las tuplas con los atributos de las tablas.

A continuación se detallan cada uno de los pasos para la carga de datos, y en el caso en que existan diferencias de tratamiento de los datos entre tablas se indicará en qué consisten.

3.5.1. Obtención de los ficheros de carga

En este paso se pretende obtener un fichero en formato CSV para cada tabla, donde cada fila represente una tupla de las tablas origen, y donde los valores de los atributos de cada fila estén separados por una coma. Dichos ficheros son *Uber.csv*, *PointsInterest.csv* y *Districts.csv*. La estructura de las distintas filas de los distintos ficheros sería la siguiente:

Atributo 1, Atributo 2, Atributo 3…… , Atributo (n)

Como se explicará en este paso, para la tabla *Districts* no se usa la coma como separador de valores, sí en el resto de tablas.

El proceso de obtención de los ficheros CSV difiere ligeramente para cada tabla. Por cada uno de los ficheros se describe a continuación de forma detallada los aspectos más relevantes acerca de la construcción de los mismos:

- Fichero *Uber.csv*: Se parte del fichero *Uber.csv* inicial. El formato de las filas es el que se requiere para la importación que se ha diseñado, es decir, los valores de cada una contienen los atributos de la tabla separados por comas. El único cambio que se ha realizado es eliminar la primera fila, cuyo contenido era el nombre de cada atributo separado por comas.

- Fichero *PointsInterest.csv*: Este fichero, tal como se adelantaba en el punto *3.3. Explicación específica del caso práctico*, se ha obtenido de forma manual, localizando puntos de interés de San Francisco mediante Google Maps y anotando cada uno en una fila distinta con sus atributos separados por comas (Ilustración 7). Se prestó especial atención a que cada punto obtenido del mapa estuviera dentro de los distritos que finalmente se incluyeron en la tabla *Districts* del modelo de datos. Los campos de cada fila son: el nombre, el tipo de localización, la latitud y la longitud.

	A	B	C	D	E	F	G
1	Paul Revere Elementary School,centro de estudios,37.736683,-122.412681						
2	Bernal Heigths Park,parque,37.743116,-122.414858						
3	Vega,restaurante,37.739307,-122.417428						
4	Holy Water,pub,37.739355,-122.418262						
5	Willie Woo Woo Wong Playground,parque,37.793503,-122.407187						
6	Far East Cafe,restaurante,37.792956,-122.406382						
7	Chinatown Merchantas Association,gubernamental,37.793329,-122.406050						
8	Transamerica Pyramid,monumento,37.795181,-122.402809						

Ilustración 7: Contenido de *PointsInterest.csv*

- Fichero *Districts.csv:* En el punto *3.3. Explicación específica del caso práctico* se indicaba que los datos pertenecientes a los distritos de San Francisco se obtuvieron del fichero *districts.xlsx*. Este fichero contiene datos de veintidós distritos de San Francisco, en concreto, su nombre y sus coordenadas geográficas. De esos veintidós distritos, se decidió eliminar dos de ellos, Twin Peaks y Downtown, el primero por errores en sus coordenadas definitorias, y el segundo por agrupar a su vez a otros distritos.

  La estructura final de las tuplas difiere de la presentada en la tabla *districts.xlsx*, donde finalmente se eliminó la primera entrada perteneciente al nombre de las columnas y se adaptó el formato de la geometría para poder crear polígonos con la función espacial *GeomFromText* de Hive (*2.3.3. Funciones espaciales en PostGIS y Hive*), tal como se adelantaba en el punto *3.4. Creación de base de base de datos*. Dicha geometría, el campo *geometryD* de la tabla, ha sido diseñada específicamente para el uso de esta función. Este campo consta de la descripción en formato *Well Known Text* (WKT) del polígono de cuatro lados iguales dos a dos que aproxima el perímetro del distrito. En *3.3. Explicación específica del caso práctico* se explicaba que para este caso práctico se utilizaban aproximaciones perimetrales de los distritos de San Francisco creadas a partir de las coordenadas de *districts.xlsx*. El resultado de esta aproximación es un rectángulo cuyos vértices son las cuatro combinaciones posibles entre latitudes norte/sur y longitudes oeste/este que vienen especificadas en la tabla de *districts.xlsx*.

  En la Ilustración 8, se puede ver como ejemplo la descripción inicial de la que se partía del distrito Bernal Heights en el fichero *districts.xlsx*.

	A	B	C	D	E
1	District	North	South	East	West
2	Bernal Heights	37,748422	37,732727	-122,405977	-122,419925

Ilustración 8: Descripción de Bernal Heights en la tabla *districts.xlsx*

En comparación con esa descripción inicial, se muestra a continuación cómo ha resultado ser el formato final de cada tupla de los distritos en *Districts.csv*:

Bernal Heights_POLYGON((-122.405977 37.732727,
-122.419925 37.732727,
-122.419925 37.748422,
-122.405977 37.748422,
-122.405977 37.732727))

Como se indicaba antes, el polígono del distrito está formado por la combinación de las coordenadas geográficas de la tabla *districts.xlsx*. El primer punto y el último son el mismo, ya que los polígonos en formato WKT deben de definirse de manera completa, es decir, hay que indicar dos veces uno de sus puntos para que se identifique el polígono como tal, una línea cerrada con mismo punto de inicio y fin.

El último aspecto a aclarar sobre el diseño de tuplas en *Districts.csv* es la elección de un carácter separador distinto a la coma, en este caso la barra baja (_). La elección de este carácter se debe a los problemas que conlleva la constitución de la expresión regular que se ha usado para cargar los datos en las tablas de Hive. El carácter coma se usa en la definición del polígono en formato WKT, haciendo que la expresión regular tomara las comas del polígono como separadores de campo. Por este motivo, se decidió cambiar la coma por la barra baja para separar los campos de la tabla. Estas expresiones regulares son explicadas en el segundo paso de la carga de datos, punto desarrollado a partir del siguiente epígrafe.

3.5.2. Carga de los datos en Hadoop/Hive

Los ficheros de carga de datos, para disponer de ellos en Hive, deben de estar localizados dentro del sistema de ficheros de Hadoop. Con este fin, se debe utilizar la consola de Hadoop para realizar la operación de copia. Para acceder a la consola de Hadoop se debe ejecutar el archivo *bat* llamado *Hadoop.bat* (ver *Anexo III*). Una vez iniciado Hadoop, se usa el comando de Hadoop *CopyFromLocal* para copiar los ficheros desde una ruta local a una ruta dentro de HDFS. Un ejemplo de comando, el cual se ha utilizado para copiar el fichero *Uber.csv*, sería el siguiente:

hadoop fs -copyFromLocal C:\Users\Administrator\Desktop\Uber.csv hdfs:///user/hadoop/

La directiva *fs* viene de "*file system"*, y engloba todo un conjunto de comandos de Hadoop para manejar ficheros en HDFS (*Hadoop File System*). Entre estos comandos se encuentra *CopyFromLocal*. Este comando copia el contenido indicado por la ruta del primer argumento dentro de la ruta de HDFS indicada por el segundo argumento. En el ejemplo anterior se copiaría *Uber.csv* desde el Escritorio del usuario Administrador dentro de la carpeta *user/hadoop/* de HDFS. Para ver que se ha copiado correctamente el fichero se puede utilizar el comando *ls*, también incluido dentro del grupo *fs*. El comando de Hadoop *ls* (*list*) muestra por pantalla de línea de comandos el contenido de un directorio indicado como argumento. Muestra datos como el tamaño, permisos, usuario propietario, fecha de creación o ubicación. Si se quiere comprobar si se han copiado correctamente los ficheros CSV, se puede usar este comando de la siguiente forma:

hadoop fs -ls /user/hadoop/

Al lanzar este comando después de haber copiado los tres ficheros en Hadoop, estos deben aparecer listados (Ilustración 9).

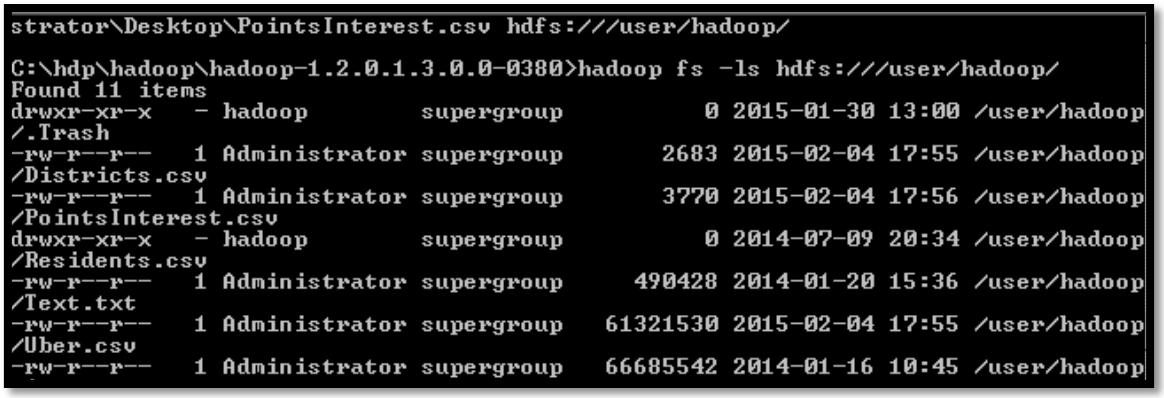

Ilustración 9: Ficheros CSV cargados en Hadoop

Cabe decir que si se apaga la máquina virtual de Windows Server sobre la que se está trabajando se deberán copiar de nuevo los ficheros dentro de HDFS, ya que este sistema de ficheros se genera en tiempo real, por lo que los ficheros estarán disponibles durante la ejecución del sistema operativo, pero no si se apaga y se vuelve a iniciar.

Una vez que se disponga en Hadoop de los tres ficheros, se pueden cargar los datos en las tablas a partir de ellos. Para realizar este último paso, se ha utilizado como base el ejemplo proporcionado por Hortonworks en su web para realizar importaciones en Hive mediante ficheros CSV (*Comma Separated Value)* (*7. Bibliografía, Otras fuentes de interés*). Este ejemplo muestra cómo cargar los datos de un fichero CSV en una tabla de Hive en dos sencillos pasos. El primero de ellos consiste en cargar todas las filas en una tabla intermedia y en una sola columna de tipo texto (*string* en Hive), y el segundo paso es descomponer esa columna haciendo uso de una expresión regular e insertar cada valor en la columna que le pertenece. ¿Por qué importar así? El fichero *Uber.csv* tiene más de un millón de registros, por lo que insertar una a una las filas no es un método práctico y muy ineficiente. Para estandarizar la importación se decidió que las otras dos tablas cargarían sus datos de igual manera que *Uber*, pese a que por ejemplo *Districts* tiene sólo veinte tuplas.

Como se ha explicado durante este punto, la importación se hace en dos pasos. El primer paso es cargar los datos dentro de tablas intermedias con una sola columna. Esta columna es de tipo *string*, y almacena todo el contenido de cada fila de fichero CSV nativo. Para cargar los datos de cada fichero CSV en la tabla intermedia se usa el comando LOAD DATA de Hive. En el ejemplo siguiente puede verse su utilización más común, usada en esta práctica, donde a partir de un archivo CSV localizado en una determinada de Hadoop se cogen los datos y se insertan uno a uno en la tabla intermedia.

LOAD DATA INPATH 'hdfs:///user/hadoop/Uber.csv' OVERWRITE INTO TABLE uber_temp;

El comando INPATH indica que se cogerán los datos del fichero contenido en la ruta especificada por el primer argumento, y se cargarán en la tabla especificada en el segundo argumento. Pueden indicarse condiciones de inserción, como la que aparece en este ejemplo, OVERWRITE, para indicar en este caso que se sobrescriben los datos que hubiera previamente en la tabla *Uber*.

Tras esta operación ya se dispone de los datos en Hive. El último paso es copiar cada tupla dentro de su tabla pertinente: *Uber, PointsInterest, Districts.* Para conseguir esto, se hace uso de expresiones regulares dentro de instrucciones SQL de selección. El objetivo es descomponer las filas, que están en formato texto, en campos e insertar cada uno de ellos en la columna de la tabla que le corresponda. El siguiente ejemplo muestra la expresión regular utilizada para cargar los datos de *Uber* desde la tabla temporal *uber_temp* .

```
INSERT OVERWRITE TABLE Uber
SELECT
regexp_extract(col_temp, '^(?:([^,]*)\,?){1}', 1) idU,
regexp_extract(col_temp, '^(?:([^,]*)\,?){2}', 1) datetimeU,
regexp_extract(col_temp, '^(?:([^,]*)\,?){3}', 1) latitudeU,
regexp_extract(col_temp, '^(?:([^,]*)\,?){4}', 1) longitudeU
FROM uber_temp;
```

El INSERT de inserción incluye un SELECT con cuatro extracciones sobre la única columna de la tabla *uber_temp*. Sobre las cuatro selecciones actúa una expresión regular sobre las tuplas que guarda *col_temp*. La expresión regular diseñada para cada uno de los campos consiste en usar las comas como medio de separación de los mismos. La primera parte de la expresión regular, *([^,]*),* indica que se recorrerá el *string* hasta que se encuentre una coma. Las cifras ubicadas entre llaves, *{1}* mapean el número del valor que se ha obtenido hasta antes de la coma, y lo señalan, en este caso, como el primer atributo de la tabla. Es decir, si el *IdU* es el primer valor de las tupla, el valor de *IdU* se insertará en la columna *IdU* de *Uber*, la primera columna de la tabla. El resto de atributos tienen el mismo tratamiento.

El ejemplo anterior se ha usado tanto para *Uber* como para *PointsInterest*, pero no para *Districts*. El fichero de carga de *Districts* se diseñó para que no hubiera conflicto con los caracteres separadores en la expresión regular de importación. Se usó una barra baja en vez de una coma para separar los dos campos que tienen, nombre y geometría. La instrucción de inserción es muy similar, como puede verse a continuación.

```
INSERT OVERWRITE TABLE Districts
SELECT
regexp_extract(col_temp, '^(?:([^_]*)\_?){1}', 1) nameD,
regexp_extract(col_temp, '^(?:([^_]*)\_?){2}', 1) geometryD
FROM districts_temp;
```

El único cambio que tuvo que hacerse en las expresiones regulares para *Districts* fue cambiar la barra baja por la coma. Es decir, en la expresión regular de búsqueda se cambia la coma y quedaría *([^_]*) y no ([^,]*).* También es necesario cambiar *\,?* por *_?*. Con estos cambios se toma la barra baja como carácter separador de campos.

Haciendo uso de estas inserciones con expresiones regulares en las tres tablas ya se dispondrá finalmente de todo el *data set* de trabajo para este caso práctico. El paso final es ejecutar cada expresión regular en la consola de Hive. Para comprobar que se han insertado correctamente los datos, se realiza una consulta sencilla en Hive sobre la tabla *Uber* por ejemplo, como la del ejemplo siguiente. Con esta consulta se obtienen todos atributos de los cinco primeros registros de la tabla.

*SELECT * from Uber limit 5;*

Si se ejecuta la consulta en Hive se obtendría el resultado de la Ilustración 10. Ya se dispone de todo lo necesario para trabajar sobre el modelo de datos.

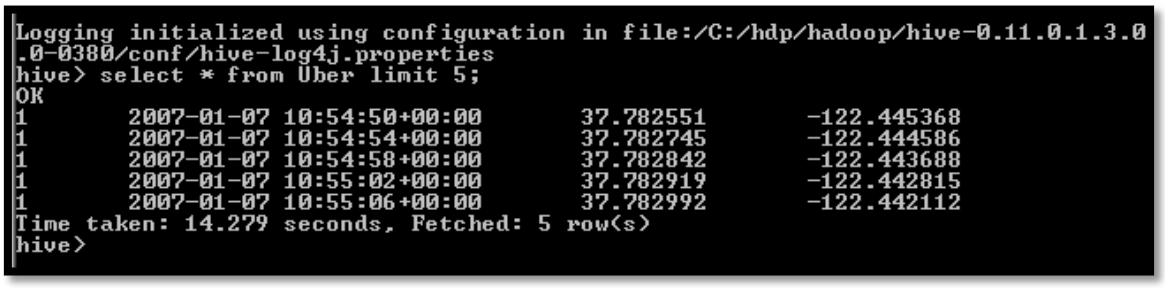

Ilustración 10: Comprobación de carga de datos correcta en *Uber*

3.6. Generación asistida de consultas espaciales

La experimentación en Hive mediante el diseño, implementación y prueba de consultas espaciales permite identificar las distintas formas unívocas que existen para diseñar una búsqueda de este tipo. A lo largo de este apartado se explica de qué forma identificar los posibles caminos de implementación y cómo llegar a obtener una consulta completa que funcione. Para ello, se ha creado y detallado con ese fin un esquema de decisiones (Ilustración 11). Con este esquema y su respectiva guía explicativa basada en ejemplos cualquier usuario iniciado en Hive podrá comprender cómo realizar búsquedas sobre datos espaciales de manera sencilla, entendiendo las decisiones de implementación y las limitaciones del lenguaje frente a un SQL estándar como el de PostGres.

Este esquema se debe entender siempre como un primer contacto con el lenguaje HQL y Hive. Para el correcto entendimiento del mismo y de los ejemplos aquí expuestos, no obstante, es recomendable conocer los fundamentos básicos de SQL, ya que HQL en sintaxis y términos técnicos imita el estándar.

El esquema de consultas representa las decisiones más genéricas de diseño de una consulta en Hive, teniendo en cuenta los siguientes factores, determinantes a la hora de confeccionar los distintos caminos de implementación:

- Patrones de diseño: La clasificación más genérica se realiza utilizando los tres patrones de diseño base de los que se parte, y que se corresponden con el principal elemento

diferenciador entre tipos de consultas. Se corresponde con las decisiones 1 y 2 del esquema.

- Elementos espaciales: El uso de funciones y datos espaciales hace que existan diferencias con respecto a no usarlas. La problemática en este caso está enfocada en la unión de tablas, donde existen diversos factores que impiden hacer esta operación de la misma manera que se haría en un entorno SQL como PostGres. Se corresponde sobre todo con la decisión 3, pero en general es el principal factor de conflicto cuando se diseña una consulta espacial, por lo que está presente en todas las ramas.

- Elementos HQL: Con respecto al SQL estándar, y pese a estar concebido como un símil de éste, *Hive Query Language* no es tan usable. Existen diferencias notables de implementación entre el SQL de PostGres y el de Hive, centradas sobre todo en cómo se estructuran las cláusulas y cómo afectan a los atributos que participan en ellas. HQL en este sentido está más limitado, tal como se verá en esta memoria. Este punto se corresponde con las decisiones 4 y 5 en el esquema de implementación.

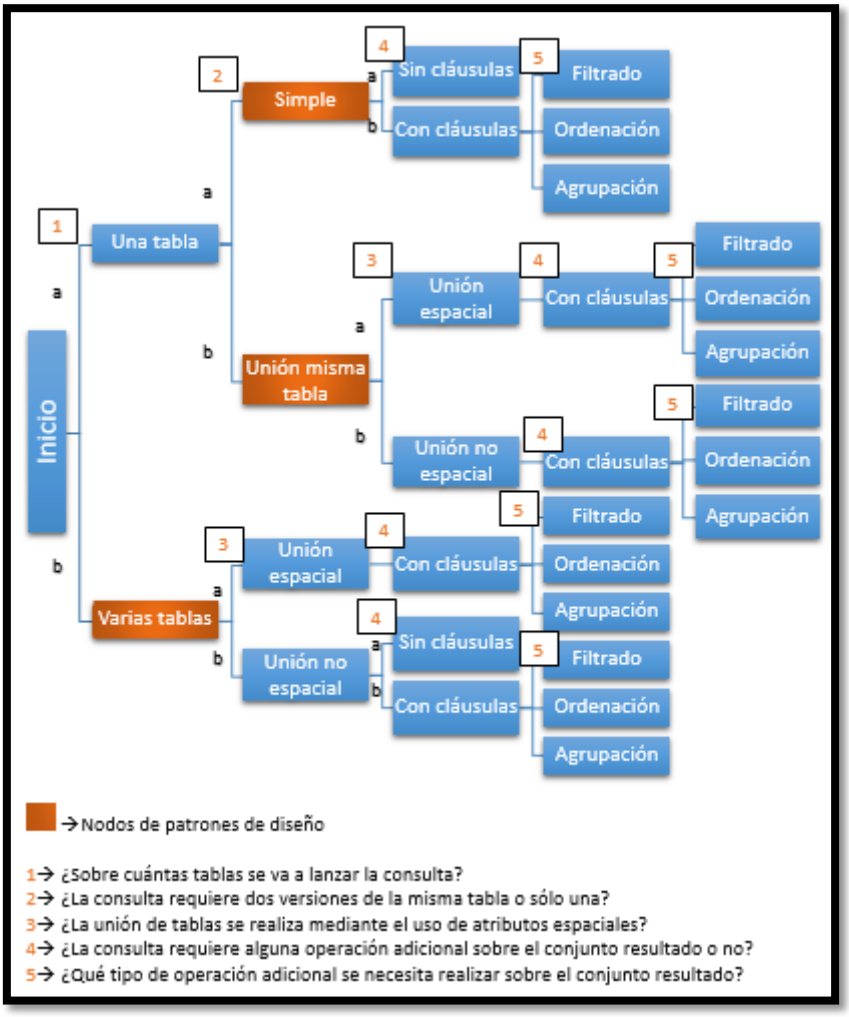

Ilustración 11: Esquema de diseño de consultas y decisiones

Cada una de las decisiones de diseño mostradas en el esquema tiene su finalidad:

- Decisión 1: Elección básica. Se debe decidir si los datos que se necesitan se pueden recuperar operando sobre una misma tabla, o si por el contrario hay que trabajar sobre más de una. Es importante recalcar que no es el mismo concepto unir tablas que consultar sobre tablas distintas, ya que también se pueden unir tablas iguales.

- Decisión 2: En caso de necesitar datos sólo de una tabla es preciso ver qué necesidades tengo sobre ella. Si necesito obtener un conjunto resultado directo sobre los datos de la tabla se trata de una consulta simple; si por el contrario se necesitase comparar dos versiones distintas de la misma tabla para unirlas y escoger las filas que cumplan con parámetros comunes entre ambas muestras, se elegirá hacer una unión entre varias versiones de la misma tabla.

- Decisión 3: Tanto si se ha elegido combinar varias tablas distintas como si se unen versiones iguales de una, hay que decidir qué condiciones existen para juntar las filas. Si la condición incluye el uso de datos y funciones espaciales se deben de tener en cuenta ciertas características del lenguaje HQL para establecer las condiciones de unión; si por el contrario no intervienen elementos espaciales en las condiciones, la unión será más trivial. En este último caso, se puede tanto establecer condiciones comparativas para atributos de forma individual o entre atributos de las tablas.

- Decisión 4: Si se requiere manejar los datos para transformar el conjunto resultado final se han de utilizar directivas adicionales para obtener los registros resultantes de la manera deseada. Si no se incluye ninguna cláusula adicional el conjunto resultado consistirá en aplicar la selección de atributos (con o sin funciones aplicadas) y uniones en el caso que las haya. Si se incluyen cláusulas adicionales la consulta gana en complejidad. En ocasiones no será necesario elegir si usar cláusulas o no, puesto que su uso será obligado para obtener búsquedas útiles.

- Decisión 5: Si se decide incluir cláusulas hay que pensar qué se necesita sobre los datos que se extraen. Las operaciones más comunes son el filtrado, la agrupación y la ordenación, pudiéndose usar sólo de un tipo o combinar varias de ellas En el esquema no se han diferenciado caminos para esta decisión, puesto que no son únicos y puede haber combinación entre cláusulas.

A continuación se detallan todas las vías de diseño representadas en el esquema. Se explicarán los aspectos imprescindibles para buscar información de todas las maneras posibles, usando las tres tablas que se han diseñado, creado y rellenado para este caso práctico. Cada rama del esquema se especifica con la siguiente información:

- Objetivo: Se explica el sentido que tiene la elección de seguir esta rama.

- Estructura: Se detalla el esquema de diseño base del tipo de consultas de la rama.

- Diseño y datos de entrada y salida de las consultas de ejemplo: Se detalla cómo están implementadas en HiveQL y las decisiones de diseño, además de los datos que usa para calcular la búsqueda y aquellos que resultan de sus operaciones. Se explican las funciones espaciales utilizadas así como los tipos de datos geográficos que se manejan.

Para facilitar su lectura, las consultas mostradas en esta memoria se identifican mediante una clave alfanumérica. La clave se corresponde con la combinación de números (**decisiones**) y letras (**camino)**, precedido del **número de consulta** dentro de la rama. La rama queda identificada por la combinación de decisiones y caminos. El esquema de identificadores de consulta se puede ver en la Ilustración 12.

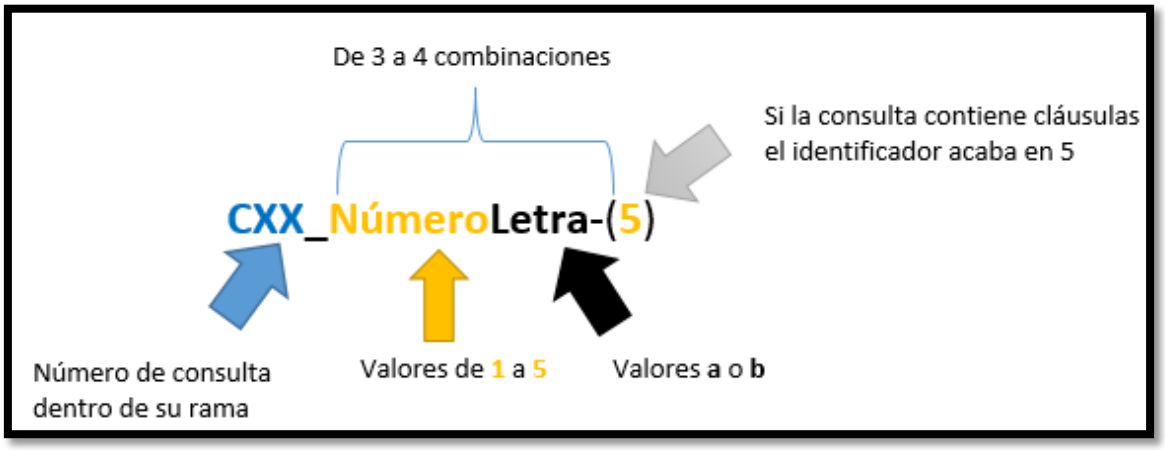

Ilustración 12: Esquema de identificación de consultas

De esta forma si se tiene una rama que consista en haber elegido realizar una búsqueda *Sobre una tabla – Simple – Sin cláusulas* dicha rama del esquema se identificaría como **1a-2a-4a**. Si es la primera consulta que se muestra de ejemplo sobre esa rama, el identificador completo de la búsqueda quedaría como **C01_1a-2a-4a**. Otro caso, si la búsqueda consiste en construir la consulta como *Sobre una tabla – Simple – Con cláusulas* la clave será **C01_1a-2a-4b-5**.

Las consultas de ejemplo se muestran en tablas, con el objetivo de facilitar su búsqueda. Las tablas tienen la estructura mstrada en la Tabla 6.

Tabla 6: Tabla esqueleto de consultas

ID	**Identificador de la consulta**
Tarjet	***"Objetivo de la consulta"***
Consulta	*Texto de la consulta*
Resultado	*Valores devueltos por la búsqueda*

Los puntos de este apartado de la memoria están pensados para seguirse en orden. No obstante, se puede consultar cualquier rama asíncronamente, ya que si existe algún aspecto de una rama o consulta no detallado éste se habrá referenciado o bien se habrá indicado dónde se desarrolla con más detalle.

EL apartado incluye el diseño e implementación de consultas, así como los resultados esperados. Las pruebas de algunos de los ejemplos más notorios aquí mostrados se incluyen en el punto *3.7. Pruebas de consultas espaciales*.

3.6.1. Una tabla – Simple – Sin cláusulas (1a-2a-4a)

El objetivo de estas consultas es extraer información desde una única tabla de datos aplicando únicamente selecciones de atributos con o sin funciones aplicadas sobre ellos. Son las consultas más básicas. El esquema canónico SQL de este tipo de consulta se corresponde con el indicado en la Ilustración 13. En estas búsquedas se requiere seleccionar mediante el comando SELECT uno, varios o todos los atributos de una tabla (A1, A2, etc), obteniendo directamente su valor, o bien aplicándoles funciones espaciales o funciones de agregación.

```
SELECT A1|Function(A1), A2|Function(A2)… An|Function(An)
FROM TablaA
```

Ilustración 13: Esquema canónico para rama 1a-2a-4a

Sobre tabla Uber

Las dos consultas que se muestran a continuación hacen uso o bien de funciones de agregación o bien de funciones espaciales. La primera consulta de ejemplo (Tabla 7) devuelve la geometría de cada punto registrado en la tabla *Uber*. Sirve para comprobar de manera visual que todas las filas de la tabla contienen un punto válido como geometría.

Tabla 7: Consulta C01_1a-2a-4a

ID	**C01_1a-2a-4a**
Tarjet	***"Geometría de cada punto de Uber"***
Consulta	***SELECT ST_GeometryType(ST_Point(Uber.longitudeU, Uber.latitudeU)) FROM Uber;***
Resultado	Devuelve la cadena de texto "ST_POINT" por cada una de las filas de Uber.

La consulta selecciona los valores de la longitud y la latitud registrados en la tupla, crea un punto geométrico con dichos datos y se devuelve el tipo de geometría. Como se puede ver, en este caso no se seleccionan atributos directamente, si no que se devuelve el valor de una función espacial. Se hace uso de dos funciones espaciales:

- *ST_Point*: Devuelve un punto geométrico en binario dadas dos posiciones X e Y. En este caso, el eje X viene indicado con la longitud espacial y el eje Y por la latitud espacial. Se usa en la SELECT para obtener un punto geométrico.

- *ST_GeometryType*: Devuelve un valor textual con el tipo de la geometría que se le pasa por parámetro, que en este caso se corresponde por el punto calculado por la función *ST_Point*.

Esta forma de comprobar que todas las tuplas contengan puntos es más correcta cuando son pocos registros con los que se trabaja, pero en este caso se trata de más de un millón de ellos. La siguiente consulta (Tabla 8) realiza la misma función que la anterior búsqueda, pero de forma más clara. Consiste en obtener el número total de puntos de la tabla *Uber*. La cifra resultante debe coincidir con el total de registros de *Uber*, en total 1.128.663 filas.

Tabla 8: Consulta C02_1a-2a-4a

ID	**C02_1a-2a-4a**
Tarjet	***"Número total de puntos registrados con Uber"***
Consulta	***SELECT COUNT (ST_Point (longitudeU,latitudeU))*** ***FROM Uber;***
Resultado	1128663

En este caso, por cada fila primero se obtiene el punto geométrico, y acto seguido se aplica una función de agregación sobre él, COUNT, la cual suma uno por cada punto que se crea con *ST_Point*, devolviéndose finalmente un único valor con el número de puntos.

NOTA: Una diferencia notable de HQL respecto a un SQL Estándar como el de PostGres es que no se pueden ejecutar instrucciones sin el comando FROM, o dicho de otra forma, siempre se ejecuta instrucciones respecto de una tabla. Por lo tanto, en Hive no se pueden ejecutar instrucciones como la siguiente:

SELECT ST_GeometryType(ST_Point (-122.405977, 37.732727));

Si se ejecuta la instrucción anterior Hive emitirá una excepción al no encontrar el comando FROM. En otros sistemas relacionales tampoco puede realizarse este tipo de selecciones.

3.6.2. Una tabla – Simple – Con cláusulas (1a-2a-4b-5)

El objetivo de estas consultas es extraer información desde una única tabla de datos aplicando cláusulas. Las cláusulas de HQL abren un abanico de posibilidades muy amplio sobre el resultado final obtenido con las búsquedas, por lo que son las ramas en las que se presentarán más ejemplos de búsqueda. El esquema canónico SQL de este tipo de consulta se corresponde con el indicado en la Ilustración 14. En estas búsquedas se requiere seleccionar con el comando SELECT uno, varios o todos los atributos de una tabla (A1, A2, etc), obteniendo directamente su valor, o bien aplicándoles a estos funciones espaciales o funciones de agregación. A

diferencia con el caso base, adicionalmente se utilizan cláusulas para ordenar, filtrar o agrupar el resultado como el usuario quiera.

```
SELECT A1|Function(A1), A2|Fun(A2)… An|Function(An)
FROM TablaA
[WHERE TablaA.A1|Function(TablaA.A1) [AND|OR TablaA.A2|
Function(TablaA.A2)]… [AND|OR TablaA.An| Function(TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun(TablaA.atribute/s)
[HAVING COUNT(TablaA.atribute)(>|=)value)]]
[ORDER BY TablaA.atribute|Function(TablaA.atribute)]
```

Ilustración 14: Esquema canónico para rama 1a-2a-4b-5

En el esquema canónico aparecen las cláusulas más utilizadas en HQL y en general en SQL para manipular los datos seleccionados en las consultas: para filtrado la cláusula WHERE y HAVING; para agrupación las cláusulas GROUP BY; y para ordenación la cláusula ORDER BY. Todas ellas son de uso opcional.

Sobre tabla Uber

Los primeros ejemplos sobre esta tabla muestran casos de filtrado. La primera consulta de ejemplo (Tabla 9) sigue el hilo de las mostradas en la categoría anterior. Se busca verificar que todas las tuplas de la tabla *Uber* contienen puntos geométricos, para lo que se hace uso de la cláusula WHERE. En este caso, con la función de agregación COUNT se obtiene una cifra que contiene el sumatorio de las filas de *Uber* que cumplen la condición del filtro WHERE, la cual consiste en que los valores de longitud y latitud proporcionados formen un punto. Un detalle es que esta vez se usa un asterisco dentro de la función COUNT, lo cual indica que se incluyen todos los atributos de la tabla en la selección para contar.

Tabla 9: Consulta C01_1a-2a-4b-5

ID	**C01_1a-2a-4b-5**
Tarjet	***"Número total de puntos registrados con Uber (conteo completo de fila)"***
Consulta	***SELECT COUNT(*)*** ***FROM Uber*** ***WHERE ST_GeometryType(ST_Point(Uber.longitudeU, Uber.latitudeU)) = 'ST_POINT';***
Resultado	1128663

NOTA: A diferencia de otros estándares como PostGres u Oracle los atributos usados en la cláusula WHERE en Hive para filtrar deben de haber sido seleccionados en la cláusula SELECT obligatoriamente. En **C01_1a-2a-4b-5** no existe este problema porque se seleccionan todos con

COUNT, pero en el ejemplo siguiente no se hace selección de los parámetros por los que se filtra:

```
SELECT IdU FROM Uber WHERE ST_Point (Uber.longitudeU, Uber.latitudeU) =
'ST_POINT';
```

En este caso el compilador de Hive dará como válida la estructura de la *query* y empezará a ejecutar, pero cuando inicie las operaciones de *Map Reduce* se emitirá un excepción *NullPointerException* de Java.

Una variante más eficiente de esta consulta es pasarle a la función de agregación COUNT únicamente el punto calculado por cada fila, evitando así la lectura de los atributos del identificador y la fecha/hora, ganando en eficiencia. Una manera de hacer esto es tener la SELECT del punto en una sub consulta, y sobre ella aplicar el COUNT y la cláusula WHERE para hacer el filtrado por el tipo de geometría. Debe de darse un alias al resultado de las funciones espaciales, que en este caso es "Punto", además de proporcionárselo también a la sub consulta, en este caso denominada como "A". El uso de alias para identar sub consultas es obligatorio, al igual que sucede en otros SQL como el de PostGres u Oracle. La Tabla 10 recoge esta alternativa de diseño.

Tabla 10: Consulta C02_1a-2a-4b-5

ID	**C02_1a-2a-4b-5**
Tarjet	***"Número total de puntos registrados con Uber (contar puntos)"***
Consulta	***SELECT COUNT (Punto)*** ***FROM (*** ***SELECT ST_GeometryType(ST_Point(Uber.longitudeU, Uber.latitudeU)) Punto*** ***FROM Uber)A*** ***WHERE A.Punto = 'ST_POINT';***
Resultado	1128663

Una búsqueda que podría ser muy común en la tabla *Uber* es encontrar la ruta o rutas que pasan por un punto determinado. La Tabla 11 contiene una *query* que devuelve el *IdU* de las rutas coincidentes con un punto concreto, en este caso, "longitud =-122.423438, latitud = 37.757945". Para evaluarlo con cada punto de *Uber*, se ha utilizado una función espacial nueva:

- *ST_Equals*: Devuelve verdadero ("TRUE") si las dos formas geométricas pasadas por parámetro son exactamente la misma, o falso ("FALSE") si no lo son.

La evaluación de la cláusula WHERE es sencilla. Si *ST_Equals* devuelve "TRUE" se selecciona el identificador de la ruta, y si no se ignora.

A partir de este ejemplo se verá en repetidas ocasiones el uso de la función espacial siguiente:

- *ST_SetSRID:* Fuerza a que una determinada geometría adquiera un SRID concreto, adjuntando el identificador espacial a los metadatos de dicha geometría.

Como se ve en la consulta de la Tabla 11, esta función utiliza sobre los dos puntos que se evalúan con *ST_Equals*. Esta función se utiliza para asegurar que ambos puntos pertenecen al mismo sistema de referencia geoespacial, que para este proyecto es el WGS84, cuyas referencias deben indicarse mediante latitudes comprendidas entre -90 y 90, y longitudes entre -180 y 180. El identificador para este sistema de referencia es el 4326.

Tabla 11: Consulta C03_1a-2a-4b-5

ID	**C03_1a-2a-4b-5**
Tarjet	***"Rutas que pasan por un punto dado"***
Consulta	***SELECT IdU*** ***FROM Uber*** ***WHERE ST_Equals(ST_SetSRID(ST_Point(longitudeU, latitudeU),4326),ST_SetSRID(ST_Point(-122.423438,37.757945),4326)) = true;***
Resultado	17711

Ahora se muestra un ejemplo con cláusula de agrupación. Se supone el caso en que se necesite obtener el número de puntos geográficos que hay registrados por cada ruta, entendiendo por ruta cada uno de los *IdU* distintos que existen en *Uber*. Para este caso, se deben agrupar las filas por el *IdU*, y seleccionar de la tabla resultante el propio *IdU* y el sumatorio de puntos por cada *IdU*. En la tabla 12 puede verse esta implementación.

Tabla 12: Consulta C04_1a-2a-4b-5

ID	**C04_1a-2a-4b-5**
Tarjet	***"Número de puntos registrado por ruta"***
Consulta	***SELECT idU,COUNT (ST_Point(Uber.longitudeU, Uber.latitudeU))*** ***FROM Uber*** ***GROUP BY idU;***
Resultado	Veinticinco mil pares *IdU*-Número de puntos por ruta. Un ejemplo de los cinco primeros resultados y los cinco últimos: 1 27 2 18 3 18 4 24 5 43 - - - - - - 24996 40 24997 38 24998 100 24999 24 25000 42

En la consulta **C04_1a-2a-4b-5** se ve otro ejemplo más sobre cómo pasarle argumentos espaciales a la función COUNT. Esta vez se hace la invocación a las funciones espaciales dentro del argumento del COUNT. Finalmente, la agrupación mediante el *IdU* se realiza con la cláusula GROUP BY. Al igual que en otros SQL estándar como el de PostGres u Oracle, el GROUP BY de Hive necesita que en la SELECT aparezcan los atributos por los que se agrupa, tal como se puede ver con *IdU* en esta consulta.

Se puede afinar más la búsqueda de **C04_1a-2a-4b-5**. Un ejemplo es ordenar los resultados por el número de puntos de cada ruta. Para ordenar, se usa la cláusula ORDER BY de HQL. Su uso es idéntico al de un SQL estándar. La forma correcta de aplicar el ORDER BY para este caso concreto es tomar **C04_1a-2a-4b-5** como una sub consulta de la ordenación, tal como se ve en la Tabla 13. Se le da un alias a la cifra que va calculando el COUNT, en este caso *puntos*, y otro a la sub consulta, denominada como *total*. Desde la consulta general se accede al resultado del conteo tanto en la SELECT como en el ORDER BY. El sentido de ordenación de ORDER BY por defecto es ascendente, por lo que se le indica mediante la directiva DESC que se quiere orden descendente. El resultado esperado es una lista de *IdU* junto con el número de puntos cada uno de ellos, ordenados de mayor a menor número de puntos.

Tabla 13: Consulta C05_1a-2a-4b-5

ID	**C05_1a-2a-4b-5**
Tarjet	***"Número de puntos registrado por ruta ordenados de mayor a menor cantidad"***
Consulta	***SELECT IdU, total.puntos*** ***FROM(*** ***SELECT idU,COUNT(ST_Point(Uber.longitudeU, Uber.latitudeU))puntos*** ***FROM Uber*** ***GROUP BY idU)total*** ***ORDER BY total.puntos desc;***
Resultado	Veinticinco mil pares *IdU*-Número de puntos por ruta, ordenados de mayor a menor Número de puntos por ruta.. Un ejemplo de los cinco primeros resultados y los cinco últimos: 1 27 2 18 3 18 4 24 5 43 - - - - - - 24996 40 24997 38 24998 100 24999 24 25000 42

NOTA: La cláusula ORDER BY de Hive no es tan usable como la de otros estándares como PostGres u Oracle. Al igual que sucede con los atributos usados con la cláusula WHERE, en Hive obligatoriamente deben de seleccionarse mediante SELECT aquellos atributos por los que se ordene con ORDER BY. Un ejemplo de uso incorrecto de ORDER BY:

SELECT ST_GeometryType(ST_Point (Uber.longitudeU, Uber.latitudeU)) FROM Uber ORDER BY Uber.datetimeU;

Al no seleccionarse *datetimeU* en la SELECT, Hive emitirá un error indicando que no encuentra ningún atributo que se llame *datetimeU*.

Como se ha visto, la cláusula WHERE permite filtrar los resultados comparándolos con cierto valor, pero no permite hacer dicho filtrado mediante el uso de funciones de agregación como COUNT. La cláusula HAVING puede usarse a modo de filtro cuando se necesita evaluar el resultado de funciones de agregación. La siguiente consulta (Tabla 14) sirve de ejemplo de uso de HAVING junto con COUNT y la función espacial *ST_Point*. Se quiere obtener aquellas rutas que tengan más de cuarenta puntos GPS registrados, en concreto, su identificador de ruta, el *IdU*. Para ello se agrupa mediante GROUP BY por el *IdU*, y se utiliza la cláusula HAVING para indicar que sólo seleccione aquellos identificadores que tengan más de cuarenta puntos geográficos. Se selecciona con SELECT en este caso únicamente el *IdU* de la tabla resultante de la agrupación y filtrado.

Tabla 14: Consulta C06_1a-2a-4b-5

ID	**C06_1a-2a-4b-5**
Tarjet	***"Rutas con más de cuarenta puntos GPS registrados con Uber"***
Consulta	***SELECT idU*** ***FROM Uber*** ***GROUP BY idU*** ***HAVING COUNT(ST_Point(Uber.longitudeU, Uber.latitudeU)) > 40;***
Resultado	*IdU* de las rutas que cumplan la condición. Un ejemplo de los cinco primeros resultados y los cinco últimos: 5 8 9 13 14 - - - 24994 24995 24996 24998 25000

Sobre tabla PointsInterest

Sobre esta tabla se detallan dos consultas relevantes de las programadas para *PointsInterest*. La primera de ellas (Tabla 15) consiste en devolver los tipos de puntos de interés de San Francisco y el número de localizaciones de cada uno de esos tipos que existen en toda la ciudad. Esta consulta es idéntica a **C04_1a-2a-4b-5**. Para obtener la cifra total de puntos de interés de cada tipo hay que agrupar mediante GROUP BY las filas por el atributo, en este caso, del tipo de localización, *typePI*. La selección consistirá en hacer la SELECT del *typePI* y de la suma de puntos geográficos, la cual se calcula con la función de agrupación COUNT.

Tabla 15: Consulta C07_1a-2a-4b-5

ID	**C07_1a-2a-4b-5**
Tarjet	***"Tipos de puntos de interés y su número de localizaciones"***
Consulta	***SELECT typePI,COUNT(ST_Point(PointsInterest.longitudePI, PointsInterest.latitudePI)) FROM PointsInterest GROUP BY typePI;***
Resultado	biblioteca 2 centro de estudios 8 edificio religioso 3 estacion 1 estadio 1 gubernamental 3 hospital 2 hotel 3 monumento 3 museo 7 parque 19 pub 6 restaurante 12 teatro 1

En el segundo ejemplo (Tabla 16) se busca encontrar el número de localizaciones de un determinado tipo que existen en San Francisco. En el caso del ejemplo se quiere conocer el número de parques de San Francisco. El resultado devuelto es un número. Para esta consulta se filtra por igualación la tabla por el tipo de localización, el *typePI*, mediante la cláusula WHERE. La selección mediante la SELECT consiste aplicar la función de agregación COUNT sobre el punto geográfico de los puntos de interés resultantes del filtrado.

Tabla 16: Consulta C08_1a-2a-4b-5

ID	**C08_1a-2a-4b-5**
Tarjet	***"Número de parques de San Francisco"***
Consulta	***SELECT COUNT(ST_Point(PointsInterest.longitudePI, PointsInterest.latitudePI)) FROM PointsInterest WHERE typePI = 'parque';***

Resultado	19

Los ejemplos sobre la tabla *PointsInterest* son muy parejos a los de la tabla *Uber* ya que las funciones espaciales usadas son las mismas, al igual que los datos espaciales, los cuales son puntos. En el siguiente grupo de consultas se expone cómo manejar polígonos.

Sobre tabla Districts

Sobre Districts en esta rama se exponen dos ejemplos sencillos de búsqueda. Los distritos están construidos como polígonos, por lo que el tratamiento es distinto con respecto a un punto. Para construir los polígonos, se utiliza una función espacial que obtiene una geometría a partir de texto plano bien formado:

- *GeomFromText*: Función que devuelve el equivalente binario de una geometría a partir de su representación en formato *Well Known Text*.

Tal como se vio en el apartado *3.4. Creación de base de datos*, la generación de polígonos en este caso práctico se ha realizado mediante el uso de esta función espacial.

Los dos ejemplos de consultas son búsquedas sencillas que podrían darse con frecuencia sobre la tabla *Districts*. Se quiere saber cuál es el distrito más grande de San Francisco y cuál el más pequeño (Tablas 17 y 18). Para conocer el tamaño de la superficie del polígono de un distrito se ha utilizado la siguiente función espacial:

- *ST_Area*: Obtiene el área de la geometría/geografía proporcionada por parámetro. Como en este caso práctico se generan geometrías para representar geografías, las unidades en las que la función devuelve el área es en grados espaciales.

La consulta está estructurada en una sub consulta y una consulta. La sub consulta recorre los distritos y calcula su área aplicando *ST_Area* sobre la geometría calculada con *ST_GemFromText*, y los encapsula en un alias. Como ya se ha explicado, ORDER BY en Hive necesita que la SELECT incluya el atributo de ordenación, por lo que además del nombre se selecciona el *área* para poder indicarlo como parámetro de ordenación. Se aplica ordenación descendente con DESC para el caso de obtener el distrito más grande, y ASC para obtener el menor, y se limita el número de elementos seleccionados a 1 con la directiva LIMIT. Por último, con la consulta genérica se selecciona de la fila seleccionada con la sub consulta sólo el nombre del distrito. Como se puede ver, el motivo de este diseño consulta-sub consulta se debe a no poder realizar ordenación mediante ORDER BY sin seleccionar, en este caso, el área de los distritos. Esta dinámica es muy repetida en las consultas que se han diseñado para Hive en este trabajo.

Tabla 17: Consulta C10_1a-2a-4b-5

ID	**C10_1a-2a-4-5**
Tarjet	***"Distrito más grande de San Francisco"***

Consulta	*SELECT b.nameD* *FROM(* *SELECT d.nameD, ST_Area(ST_GeomFromText(d.geometryD,4326)) area* *FROM Districts d* *ORDER BY area DESC* *LIMIT 1) b;*
Resultado	Sunset District

Para obtener justo el caso contrario, es decir, los distritos más pequeños, sólo hay que cambiar la directiva DESC por ASC (o no poner ninguna, ya que por defecto se aplica sentido ascendente).

Tabla 18: Consulta C11_1a-2a-4b-5

ID	**C11_1a-2a-4b-5**
Tarjet	***"Distrito más pequeño de San Francisco"***
Consulta	*SELECT b.nameD* *FROM(* *SELECT d.nameD, ST_Area(ST_GeomFromText(d.geometryD,4326)) area* *FROM Districts d* *ORDER BY area ASC* *LIMIT 1) b;*
Resultado	Chinatown

Ejemplos derivativos tan sencillos como obtener **C11_1a-2a-4b-5** a partir de **C10_1a-2a-4b-5** sirven para ver que HQL funciona de forma prácticamente idéntica a como lo hacen otros SQL en cuanto a sintaxis.

3.6.3. Una tabla–Unión misma tabla–Unión espacial – Con cláusulas (1a-2b-3a-4-5)

Hasta ahora se han visto consultas simples sobre una única tabla, donde se manipulaban o no los datos con funciones agregación o espaciales, o bien mediante cláusulas. La otra rama que se abre desde la decisión **2** también supone consultar sobre una tabla, pero jugando con dos versiones de la misma. El objetivo de los casos que se plantean para este trabajo siempre será enfrentar ambas versiones de las tablas para hacer un filtrado de una sobre la otra.

En el esquema de generación de consultas no se incluyen para las consultas de unión sobre misma tabla la opción de no usar cláusulas. Esto es debido a que el uso de cláusulas en este tipo de consultas es obligado, ya que no tiene fundamento unir dos tablas que son iguales. Tanto si la unión es espacial como si no es necesario usar las cláusulas,.

La forma canónica de este tipo de consultas puede verse en la Ilustración 15. Es una combinación de la estructura que poseían las consultas simples con cláusulas. Se tienen dos bloques fijos con los parámetros nombrados con A (dos versiones de la tabla), y adicionalmente se pueden añadir más versiones de la misma tabla (parámetros nombrados en N). Las distintas

versiones de la tabla sobre la que se trabaje se combinan mediante el comando JOIN. En resumen, lo que se tienen son varias versiones de la tabla, las cuales se unen para según la cláusula/s usadas quedarse con ciertos resultados de la misma. Las cláusulas genéricas WHERE y ON mostradas al final de la forma canónica se corresponde en este caso con el criterio de unión de las tablas, el cual debe contener una función espacial como mínimo.

```
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function(TablaA.A1) [AND|OR TablaA.A2|
Function(TablaA.A2)]… [AND|OR TablaA.An| Function(TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun(TablaA.atribute/s)
[HAVING COUNT(TablaA.atribute)(>|=)value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
JOIN
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) (>|=) value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
 [JOIN
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) (>|=)value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)] n veces
ON|WHERE TablaA.A1|Function(TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.Nn)]
```

Ilustración 15: Esquema canónico para rama 1a-2b-3a-4-5

A partir de esta rama aparece un nuevo comando de HQL, JOIN, el cual permite realizar combinaciones de tablas. Las uniones pueden realizarse de varias formas, pero la más utilizada en este trabajo práctico (por motivos que se explicarán en los ejemplo más adelante) es la combinación de JOIN con la cláusula WHERE, cuando la unión es espacial. La cláusula ON solo se ha reservado en determinados casos para cuando no se une por atributos espaciales, ya que como se verá, dificultan la implementación. En ambos casos, el nexo de unión puede ser una comparación de atributos o de atributos frente a un valor.

Como se ha visto, pueden realizarse uniones de más de dos versiones de la misma tabla, pero en esta memoria sólo se muestran ejemplo sobre dos, para facilitar la iniciación a este tipo de consultas.

Sobre tabla Uber

Como se ha adelantado en la introducción a esta rama, el uso más común que tiene la unión de dos versiones de la misma tabla es comparar y/o filtrar filas entre una y otra. El primer ejemplo que se muestra a continuación es una versión de **C03_1a-2a-4b-5**, donde se buscaba aquellas rutas que pasaban por un punto dado. A diferencia de aquella consulta simple, este ejemplo (Tabla 19) compara los puntos de una de las rutas con el resto de puntos de las demás rutas. Esto no puede hacerse con un filtrado simple como los vistos hasta ahora, se necesita tener dos versiones de la tabla para poder compararlas y seleccionar las filas que cumplan la condición.

Para esta consulta, el *modus operandi* consiste en, por un lado, seleccionar las tuplas que tengan IdU = 5, y por otro lado, seleccionar aquellas que no tengan *IdU* = 5. Es decir, se dispondrá de dos SELECT distintos. Cada SELECT recibe un alias, A para *IdU* = 5 y B para *IdU* ¡= 5. Hasta aquí, lo que se ha obtenido son dos versiones de *Uber*. Ambas se unen mediante el comando JOIN. Para elegir qué tuplas se unen y cuáles no en la unión, se utiliza la cláusula de filtrado WHERE. En dicha cláusula se indica que haga JOIN cuando los puntos de una tabla y otra sean iguales, para lo cual se usa la función ST_Equals sobre la generación de los dos puntos de ambas tablas con ST_Point. Se fuerza mediante ST_SetSRID que los puntos que proceden de una y otra selección están dados en el estándar WSG84.

Queda ver qué se selecciona del conjunto resultante. En este caso, se va a seleccionar el identificador de ruta *IdU*, el año de toma de la ruta y el punto común por el que han pasado ambas rutas, dado en formato *Well Known Text*. El año de registro del punto se obtiene aplicando la instrucción SUBSTR sobre los cuatro primeros caracteres de *DatetimeU*. El punto en común en formato WKT se obtiene aplicando la siguiente función espacial:

- ST_AsText: Devuelve una representación textual en formato Well Known Text de la geometría dada por parámetro, en este caso, el punto en común entre las dos rutas evaluadas.

ST_AsText recibe como parámetro el punto calculado por ST_Point para las coordenadas de la tabla A, es decir, para la tabla que contiene toda la ruta *IdU* = 5. El mismo resultado se obtendría si se seleccionara el punto sobre los atributos de la tabla B, ya que debe ser el mismo punto que A.

Tabla 19: Consulta C01_1a-2b-3a-4-5

ID	**C01_1a-2b-3a-4-5**
Tarjet	***"Rutas de Uber que han pasado por donde lo hizo la ruta 5"***
Consulta	***SELECT A.idU, SUBSTR (A.datetimeU,0,4), B. idU, SUBSTR(B.datetimeU,0,4), ST_AsText (ST_Point (A.longitudeU,A.latitudeU)) FROM (SELECT * FROM Uber WHERE idU = 5) A***

	JOIN (***SELECT * FROM Uber WHERE idU != 5) B*** ***WHERE*** ***ST_Equals(ST_SetSRID(ST_Point(A.longitudeU,A.latitudeU),4326),ST_SetSRID(ST_Point(B.longitudeU,B.latitudeU),4326))=true;***
Resultado	5 2007 24684 2007 POINT (-122.43854522705078 37.77860641479492) 5 2007 19742 2007 POINT (-122.43870544433594 37.779449462890625) 5 2007 19991 2007 POINT (-122.43830871582031 37.77759552001953)

NOTA: En este trabajo práctico no se han utilizado JOIN con cláusula de filtrado ON cuando la condición del mismo consiste en igualar de forma directa una función espacial a un valor, ya que usarlo conlleva el aumento de la complejidad de las consultas. En Hive no se pueden comparar de forma directa una función espacial con un valor dentro de una condición en ON. El uso de ON es una de las prácticas más eficientes para hacer JOIN, y la más común. El problema es que en HQL no es tan usable como en otros SQL como PostGres. Usando de ejemplo la consulta anterior, **C01_1a-2b-3a-4-5**, si se cambia WHERE por ON (dejando el resto de la consulta igual) se debería obtener el mismo resultado, sin embargo Hive no admite sintácticamente la condición de filtrado de *ST_Equals(point,point) = true*. Para poder hacer este tipo de JOIN/ON debe de recogerse dicho ST_Equals en un alias, para luego usarlo en el ON. Sin embargo, esto añade complejidad a la hora de implementar la consulta, puesto que se debe encapsular la función en un alias en una sub consulta y llamarla desde una consulta superior. Por este motivo, los JOIN en los que se comparan funciones espaciales con valores, como en este caso valor *true*, se han implementado con cláusula WHERE. Sólo se muestran casos, como se verá a continuación en el siguiente ejemplo, donde ON conlleva comparar el resultado de dos funciones espaciales.

La siguiente consulta (Tabla 20) es idéntica a **C01_1a-2b-3a-4-5**, con la particularidad de que el filtrado de su JOIN se hace mediante la igualación de sus puntos, no mediante ST_Equals. En la cláusula WHERE, se iguala el resultado de ST_Point para cada versión de *Uber*.

Tabla 20: Consulta C02_1a-2b-3a-4-5

ID	**C02_1a-2b-3a-4-5**
Tarjet	***"Rutas de Uber que han pasado por donde lo hizo la ruta 5" (Con igualación de puntos)"***
Consulta	***SELECT A.idU, SUBSTR (A.datetimeU,0,4), B. idU, SUBSTR(B.datetimeU,0,4), ST_AsText (ST_Point (A.longitudeU,A.latitudeU))*** ***FROM (*** ***SELECT * FROM Uber WHERE idU = 5) A*** ***JOIN (*** ***SELECT * FROM Uber WHERE idU != 5) B*** ***WHERE ST_SetSRID(ST_Point(A.longitudeU,A.latitudeU),4326) = ST_SetSRID(ST_Point(B.longitudeU,B.latitudeU),4326);***
Resultado	5 2007 24684 2007 POINT (-122.43854522705078 37.77860641479492) 5 2007 19742 2007 POINT (-122.43870544433594 37.779449462890625) 5 2007 19991 2007 POINT (-122.43830871582031 37.77759552001953)

Por último, la Tabla 21 muestra una variante sobre **C02_1a-2b-3a-4-5**. En esta se hace hincapié en lo explicado en la anterior NOTA. Se puede realizar el JOIN por igualación de puntos mediante la cláusula ON, obteniendo exactamente el mismo resultado que con **C02_1a-2b-3a-4-5**. Este tipo de JOIN/ON sí son practicables en Hive. Para hacerlo efectivo en la anterior *query*, sólo hay que sustituir la cláusula WHERE por ON, y se obtendrá una unión interna por igualación (En SQL se conoce como INNER JOIN). Se asegura mediante el uso de ST_SetSRID que se estén comparando puntos en formato WGS84 (id 4326).

Tabla 21: Consulta C03_1a-2b-3a-4-5

ID	**C03_1a-2b-3a-4-5**
Tarjet	***"Rutas de Uber que han pasado por donde lo hizo la ruta 5" (Con igualación de puntos y JOIN/ON)"***
Consulta	***SELECT A.idU, SUBSTR (A.datetimeU,0,4), B. idU, SUBSTR(B.datetimeU,0,4), ST_AsText (ST_Point (A.longitudeU,A.latitudeU)) FROM (SELECT * FROM Uber WHERE idU = 5) A JOIN (SELECT * FROM Uber WHERE idU != 5) B ON ST_SetSRID(ST_Point(A.longitudeU,A.latitudeU),4326) = ST_SetSRID(ST_Point(B.longitudeU,B.latitudeU),4326);***
Resultado	5 2007 24684 2007 POINT (-122.43854522705078 37.77860641479492) 5 2007 19991 2007 POINT (-122.43830871582031 37.77759552001953) 5 2007 19742 2007 POINT (-122.43870544433594 37.779449462890625)

3.6.4. Una tabla–Unión misma tabla–Unión no espacial–Con cláusulas(1a-2b-3b-4-5)

Esta rama busca el mismo objetivo que **1a-2b-3a-4-5**: comparar distintas versiones de una misma tabla. La principal diferencia de esta rama con **1a-2b-3a-4-5** es la no utilización de funciones espaciales para establecer los criterios de unión para el comando JOIN. Esto hace que ahora no existan restricciones a la hora establecer los criterios unión con la cláusula ON.

El esquema canónico de esta rama (Ilustración 16) es idéntico al esquema de la rama **1a-2b-3a-4-5**, sólo que con la diferencia de que no se usan funciones espaciales ni en el WHERE ni en el ON finales. En este caso, al no usarse funciones espaciales, pueden usarse cualquier tipo de comparaciones, ya sean entre atributos de las tablas con otros atributos o con valores.

```
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function(TablaA.A1) [AND|OR TablaA.A2|
Function(TablaA.A2)]… [AND|OR TablaA.An| Function(TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun(TablaA.atribute/s)
[HAVING COUNT(TablaA.atribute)(>|=)value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
JOIN
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) (>|=) value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
 [JOIN
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) (>|=)value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)] n veces
ON|WHERE TablaA.A1|Function(TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]
```

Ilustración 16: Esquema canónico para rama 1a-2b-3b-4-5

Sobre tabla PointsInterest

Para ejemplificar casos de búsqueda sobre esta tabla, se va a seguir la misma dinámica que la seguida con los ejemplos para *Uber* en **1a-2b-3a-4-5**. Se va a mostrar una consulta sobre *PointsInterest* en la cual se hace unión mediante un JOIN y filtrado del mismo con WHERE, para luego mostrar otra variante en la que se realizará la misma consulta pero usando ON en lugar de WHERE. Al no ser uniones espaciales no existen problemas de compatibilidad con Hive con esta operación, como sí pasaba con la anterior rama. En ambos casos se verá también cómo hacer una ordenación con este tipo de uniones, además de conocer el uso de una nueva función espacial no visto hasta ahora.

La Tabla 22 muestra un ejemplo en el cual se busca encontrar los cinco puntos de interés de San Francisco más cercanos a uno dado, en este caso, el Hotel Fairmont. Los resultados se devuelven ordenados de más cercanos a más lejanos del punto. Si se analiza esta *query*, se

puede ver que es muy parecida en estructura a **C01_1a-2b-3a-4-5, C02_1a-2b-3a-4-5** y **C03_1a-2b-3a-4-5**, estando formada por dos sub consultas: el lado izquierdo del JOIN selecciona todos los datos de la fila correspondiente al Hotel Fairmont, mientras que el lado derecho se corresponde con la SELECT que almacena el resto de puntos de interés sin incluir dicho Hotel. Cada parte tiene su propio filtro WHERE para seleccionar cada tipo de información: el WHERE que saca la fila de hotel consiste en igualar el *nameP* con el nombre completo, mientras que el que sirve para sacar el resto de puntos en la otra SELECT en vez de igualarlo de distingue con "¡=". Ambas partes se recogen con alias (*hotel* y *puntosInteres)*. Este JOIN debe recogerse en una consulta superior, debido a que, como ya se ha identificado en ejemplos anteriores, la ordenación mediante ORDER BY requiere que el valor por el que se ordene esté seleccionado en la SELECT. El valor por el que se ordena es una distancia calculada con la función espacial siguiente:

- ST_Distance: Calcula la distancia cartesiana existente entre dos geometrías o dos geografías. En el caso que se está tratando, la distancia calculada se corresponde con la existente entre el Hotel Fairmont y el resto de puntos.

De la unión de tablas, mediante el uso de SELECT se obtiene de la JOIN el nombre del punto y cada una de las distancias entre puntos, calculados mediante ST_Point sobre sus correspondientes coordenadas. De esta manera, mediante el uso de un alias sobre este cálculo, *distancia*, se puede usar ORDER BY con el valor calculado por ST_Distance. A su vez, toda esta consulta es una sub consulta de la SELECT general que saca el nombre *nombreP* de cada punto seleccionado. Por último, señalar que en este caso no existe WHERE genérico para la JOIN, si no que los filtros de cada SELECT a cada lado de JOIN hacen esta labor, quedándose sólo con los registros que interesan. También se limitan los resultados finales a 5 registros con la instrucción LIMIT, aplicada ya cuando se han obtenido los nombres de los puntos cercanos ordenados, es decir, sobre toda la consulta. Nótese que también se utiliza ST_SetSRID a la hora de calcular las distancias con ST_Distance para forzar que se use el sistema de referencia espacial WGS84 (id 4326).

Tabla 22: Consulta C01_1a-2b-3b-4-5

ID	**C01_1a-2b-3b-4-5**
Tarjet	***"Cinco puntos de interés más cercanos al Hotel Fairmont ordenados de mayor a menor distancia"***
Consulta	***SELECT nombreP FROM(SELECT nombreP,distancia FROM(SELECT nombreP,ST_Distance(ST_SetSRID(ST_Point(longitudePI,latitudePI),4326) ,ST_SetSRID(ST_Point(longitudeP,latitudeP),4326))distancia FROM(SELECT * FROM PointsInterest WHERE namePI = 'The Fairmont San Francisco')hotel JOIN(SELECT namePI nombreP,longitudePI longitudeP, latitudePI latitudeP FROM PointsInterest WHERE namePI != 'The Fairmont San Francisco')puntosInteres)total***

	ORDER BY distancia)total2 LIMIT 5;
Resultado	Cable Car Museum Grace Cathedral Willie Woo Woo Wong Playground Far East Café Chinatown Merchantas Association

De igual manera se puede obtener el mismo resultado que en **C01_1a-2b-3b-4-5** si se utiliza la cláusula ON en lugar de WHERE para establecer qué registros deben tenerse en cuenta al unir con JOIN. La Tabla 23 contiene una modificación de la consulta **C01_1a-2b-3b-4-5**, donde se han eliminado los filtros WHERE de las sub consultas a ambos lados de la JOIN, para así incluirlos dentro de la cláusula ON de manera conjunta, usando la directiva AND para indicar que deben cumplirse ambas dos para seleccionar las filas.

Tabla 23: Consulta C02_1a-2b-3b-4-5

ID	**C02_1a-2b-3b-4-5**
Tarjet	***"Cinco puntos más cercanos al Hotel Fairmont ordenados de mayor a menor distancia (Con JOIN/ON)"***
Consulta	***SELECT nombreP FROM(*** ***SELECT nombreP,distancia FROM(*** ***SELECT*** ***nombreP,ST_Distance(ST_SetSRID(ST_Point(longitudePI,latitudePI),4326)*** ***,ST_SetSRID(ST_Point(longitudeP,latitudeP),4326))distancia FROM (*** ***SELECT **** ***FROM PointsInterest)hotel*** ***JOIN*** ***(SELECT namePI nombreP,longitudePI longitudeP, latitudePI latitudeP*** ***FROM PointsInterest)puntosInteres*** ***ON hotel.namePI = 'The Fairmont San Francisco' AND puntosInteres.nombreP*** ***!= 'The Fairmont San Francisco')total*** ***ORDER BY distancia)total2 LIMIT 5;***
Resultado	Cable Car Museum Grace Cathedral Willie Woo Woo Wong Playground Far East Café Chinatown Merchantas Association

Sobre tabla Districts

Ahora se muestran dos ejemplos de uniones no espaciales sobre *Districts*. La estructura general de las consultas es parecida a lo visto hasta ahora en la rama **1a-2b-3b-4-5**. En ambas búsquedas se obtiene el mismo resultado, pero de manera distinta.

En las dos siguientes consultas se quiere obtener los distritos de San Francisco que limiten con Chinatown. En el primer caso (Tabla 24) se utiliza la función ST_Distance para comprobar las

distancias de Chinatown al resto de los distritos, mientras que en la segunda consulta (Tabla 25) se utiliza la siguiente función espacial:

- ST_Intersects: Devuelve verdadero si dos formas geométricas en dos dimensiones tienen algún punto en común, y falso en caso contrario.

En la primera consulta, **C03_1a-2b-3b-4-5**, se va a mostrar un ejemplo muy completo sobre cómo utilizar cláusulas de filtrado múltiple y de ordenación, en combinación con una unión sin función espacial. Una vez más, para este tipo de consulta donde se necesita comparar dos versiones de una tabla, se necesita extraer por separado los datos a comparar de cada versión de la misma. A diferencia de otras consultas anteriores como **C02_1a-2b-3b-4-5**, el lado derecho de la JOIN queda como sub consulta del lado izquierdo. Es una alternativa de diseño igual de válida que en **C02_1a-2b-3b-4-5**. Los filtros de WHERE de cada parte del JOIN seleccionan, en la parte izquierda, la fila de Chinatown, y en la derecha, todas menos Chinatown. El cálculo de la distancia en la SELECT es igual que en **C02_1a-2b-3b-4-5**, sólo que esta vez las geometrías no son puntos, sino polígonos, aunque el funcionamiento de ST_Distance es exactamente igual. Recordar el uso de ST_GeomFromText para obtener las geometrías a partir del atributo, y así usarlas en ST_Distance. En este caso, a GeomFromText se le pasa en su argumento opcional el valor 4326 en la generación de ambas geometrías, para forzar que se utilice el sistema de referencia WGS84.

Finalmente para ordenar y filtrar los resultados cuando la distancia calculada sea cero, se engloba la JOIN en dos consultas, una para cada cláusula. La primera de ellas (alias *total*) selecciona el nombre del distrito de la parte derecha de la JOIN (de aquellos no iguales a Chinatown) y la distancia (alias *distancia*), y se aplica la ordenación mediante cláusula ORDER BY sobre *distancia*. En este caso, se ha indicado orden ascendente con ASC, pero si no se indica el orden es ascendente igualmente en HiveQL. También se limitan los resultados a cinco con LIMIT. La segunda de las consultas, la más genérica (alias *completa*) engloba a todas las demás, y aplica sobre ella el filtro WHERE sobre *distancia*, donde se seleccionan los *nameD* de *completa* cuya distancia entre distritos sea igual a cero.

Tabla 24: Consulta C03_1a-2b-3b-4-5

ID	**C03_1a-2b-3b-4-5**
Tarjet	***"Distritos colindantes a Chinatown (Con ST_Distance)"***
Consulta	***SELECT completa.nombre FROM (SELECT total.nameD nombre, distancia FROM (SELECT Districts.nameD, ST_Distance(ST_GeomFromText(Districts.geometryD,4326),ST_GeomFromText(distrito.geometryD,4326)) distancia FROM Districts JOIN (SELECT * FROM Districts WHERE Districts.nameD = 'Chinatown')distrito)total WHERE total.nameD != 'Chinatown' ORDER BY distancia ASC LIMIT 5)completa WHERE completa.distancia = 0;***

Resultado	Nob Hill Financial District North Beach

Se puede resolver la anterior consulta si se utiliza, en lugar de ST_Distance, otra de las funciones espaciales para la que existe implementación en Hive: ST_Intersects. La estructura y sintaxis de la consulta son exactamente iguales que en **C03_1a-2b-3b-4-5**, sólo varía la condición que se aplica en la cláusula WHERE de filtrado de la consulta que engloba todo el conjunto, la cual comprueba que el valor devuelto por ST_Intersects sea *true*. ST_Intersects se utiliza de forma idéntica a ST_Distance, guardándose el resultado en un alias, *intersecta*, el cual es leído desde la consulta inmediatamente superior.

Tabla 25: Consulta C04_1a-2b-3b-4-5

ID	**C04_1a-2b-3b-4-5**
Tarjet	***"Distritos colindantes a Chinatown (Con ST_Intersects)"***
Consulta	***SELECT completa.nombre FROM (SELECT total.nameD nombre, intersecta FROM (SELECT Districts.nameD, ST_Intersects(ST_GeomFromText(Districts.geometryD,4326),ST_GeomFromText(distrito.geometryD,4326)) intersecta FROM Districts JOIN (SELECT * FROM Districts WHERE Districts.nameD = 'Chinatown')distrito)total WHERE total.nameD != 'Chinatown')completa WHERE completa.intersecta = true;***
Resultado	Nob Hill Financial District North Beach

3.6.5. Varias tablas – Unión espacial – Con cláusulas (1b-3a-4-5)

Las ramas que se tratan a partir de ésta buscan juntar datos de distintas tablas, con el fin de poder manejar al unísono los datos de las relaciones unidas. Es muy común que las uniones entre tablas se acompañen de cláusulas de distinto índole para potenciar la operación JOIN. En este caso, se trata de una rama donde la unión es espacial, por lo que se necesitará obligatoriamente el uso de cláusula o bien WHERE o bien ON para indicar por cuáles atributos/valores se hace la unión o bajo qué condiciones. Adicionalmente en esta rama se usan cláusulas para complementar las usadas en la unión, para obtener diversos resultados de las uniones.

Para este caso práctico sólo se han implementado uniones internas con ON (INNER JOIN) o mediante igualación con WHERE, cuando éstas se realizan usando atributos de las tablas.

El esquema canónico para esta rama se muestra en la Ilustración 17. Al contrario que en el esquema de **1a-2b-3b-4-5**, en esta rama se unen tablas distintas (A, B….N tablas). Por lo demás, las búsquedas tienen la estructura idéntica en ambas ramas. Cabe decir que al corresponderse con uniones espaciales, el ON/WHERE genérico debe contener una función espacial como mínimo.

```
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) |(>|=)value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
JOIN
SELECT B1|Function (B1), B2|Fun (B2)… Bn|Function (Bn)
FROM TablaB
[WHERE TablaB.B1|Function (TablaB.A1) [AND|OR TablaB.B2|
Function (TablaB.B2)]… [AND|OR TablaB.Bn| Function (TablaB.Bn)]]
[GROUP BY TablaB.atribute/s|Fun (TablaB.atribute/s)
[HAVING COUNT (TablaB.atribute) (>|=) value)]]
[ORDER BY TablaB.atribute|Function (TablaB.atribute)]
 [JOIN
SELECT N1|Function (N1), N2|Fun (N2)… Nn|Function (Nn)
FROM TablaA
[WHERE TablaN.N1|Function (TablaN.N1) [AND|OR TablaN.N2|
Function (TablaN.N2)]… [AND|OR TablaN.Nn| Function (TablaN.Nn)]]
[GROUP BY TablaN.atribute/s|Fun (TablaN.atribute/s)
[HAVING COUNT (TablaN.atribute) (>|=) value)]]
[ORDER BY TablaN.atribute|Function (TablaN.atribute)] n veces
ON|WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaB.B2|
Function (TablaB.B2)]… [AND|OR TablaN.Nn| Function (TablaN.Nn)]
```

Ilustración 17: Esquema canónico para rama 1b-3a-4-5

Esta rama, al contemplar la unión de varias tablas al disponer del uso de cláusulas adicionales, proporciona muchas posibilidades distintas de búsqueda. Por esta razón, los ejemplos mostrados para ella son numerosos. En muchos de los casos de prueba se trabajan uniones espaciales entre distintos tipos de datos espaciales.

Sobre tablas Uber/Districts

Las uniones de tablas cuyos atributos espaciales son distintos en tipo pueden hacerse de distintas maneras, utilizando diversos tipos de funciones espaciales. En el caso práctico que se

ha diseñado para este proyecto, la función espacial más útil que puede usarse para hacer filtro en la JOIN entre puntos y polígonos (*Uber* y *Districts* respectivamente) es la siguiente:

- ST_Contains: Devuelve valor verdadero si la segunda geometría pasada por parámetro en la función está contenida en la primera. Dicho de otra manera, al menos un punto de del segundo parámetro debe estar en el primero. Si no es así devuelve falso.

Como se verá en los ejemplos propuestos para esta rama, esta función espacial es utilizada en todos ellos, de forma única o junto con otros filtros adicionales. Centrando la atención en *Uber* y *Districts*, se van a mostrar tres ejemplos de consultas para ver cómo hacer uniones espaciales junto con el uso adicional cláusulas, más allá de las necesarias para filtrar la JOIN.

La primera consulta mostrada (Tabla 26) devuelve el número de rutas que han pasado por un determinado distrito, en este caso Sunset District. Es una *query* muy completa, ya que hace uso de varias cláusulas, además de funciones de agregación avanzadas. Esta búsqueda consta de una JOIN entre Uber (*u*) y Districts (*d*), sobre cuya tabla resultante se filtran los resultados que aparecerán mediante el uso de la función espacial ST_Contains en el filtro WHERE. A ST_Contains se le pasa como segundo parámetro el punto calculado con ST_Point sobre cada par de coordenadas de *Uber* (*longitudeU* y *latitudeU*), y como primer parámetro el polígono calculado por ST_GeomFromText sobre *geometryD* de *Districts*. El resultado calculado por ST_Contains se compara con el valor *true*, que en caso de ser igual a este valor, el punto de *Uber* se ha registrado en la zona de Sunset District. Adicionalmente se indica otra condición adicional a esta: se escoge sólo aquellas filas cuyo nombre de distrito sea Sunset District, para descartar el resto en la operación.

Hasta aquí se tiene delimitado qué datos intervienen en la consulta: puntos que pasaron por Sunset District. Como lo que interesan no son los puntos, sino el número de rutas, se debe agrupar el conjunto resultado por su *IdU* mediante la cláusula GROUP BY. Para calcular el número de rutas, se ha usado la función de agregación ROW_NUMBER( ) en la SELECT, de idéntico uso en HiveQL que en otros SQL como PostGres. Esta función complementa el GROUP BY, y si no se le indica ninguna operación adicional dentro de ROW_NUMBER ( ) o en dentro de OVER( ), devuelve el número de orden de la cada fila. El GROUP BY selecciona los *IdU* de forma distintiva, esperando que se haga alguna agregación con ellos. Pues bien, Usando ROW_NUMBER( ) en la SELECT se obtiene de fila de *IdU* su número de orden. El número de rutas se corresponde con lo que marque la última fila seleccionada, por lo que con ORDER BY se ordenan los resultados de forma descendente con la directiva DESC, y se limita además el número de resultados en uno sólo mediante LIMIT. De esta forma se obtiene el número de rutas que pasan por Sunset District. Por último, recordar el uso del estándar 4326 para las geometrías, donde en el caso del distrito se indica dentro de la función ST_GeomFromText y en el punto mediante la función ST_SetSRID.

Tabla 26: Consulta C01_1b-3a-4-5

ID	**C01_1b-3a-4-5**
Tarjet	***"Número de rutas que pasan por Sunset District"***

Consulta	*SELECT ROW_NUMBER( ) OVER( ) r* *FROM Uber u* *JOIN Districts d* *WHERE ST_Contains(ST_GeomFromText(d.geometryD,4326), ST_SetSRID(ST_Point(u.longitudeU,u.latitudeU),4326))=true AND d.nameD = 'Sunset District'* *GROUP BY u.idU* *ORDER BY r DESC LIMIT 1;*
Resultado	371

Ahora se muestran dos consultas contrarias una de la otra. Por un lado (Tabla 27) se va a buscar el nombre de los distritos que pasaron por una determinada ruta, la 53; y por otro lado (Tabla 28), se muestra un caso contrario, encontrar las rutas que pasaron pon un distrito, SOMA. Aunque este caso de búsqueda es más sencillo que **C01_1b-3a-4-5**, se podrá ver que modificando mínimamente una se obtiene otra con resultados opuestos. En ambos casos no se usan cláusulas adicionales más allá de la obligatoria que filtra la JOIN.

La primera de ellas (Tabla 27), como ya se ha adelantado, encuentra los nombres de los distritos por los que pasó la ruta con *IdU* igual 53. Es una consulta sencilla, siendo la base la misma que en **C01_1b-3a-4-5**, una JOIN entre las tablas *Uber* y *Districts*, con un filtrado WHERE genérico, donde se evalúa con ST_Contains si los puntos de *Uber* de la ruta 53 (*u.IdU* = 53) están contenidos en distritos de *Districts*, en cuyo caso se selecciona el *nameD* en la SELECT. Se fuerza, como en casos anteriores, que el sistema de referencia geográfico sea el WGS84 e la comparación de ST_Contains.

Tabla 27: Consulta C03_1b-3a-4-5

ID	**C02_1b-3a-4-5**
Tarjet	***"Distritos por los que pasó la ruta 53"***
Consulta	*SELECT DISTINCT nameD* *FROM Uber u* *JOIN Districts d* *WHERE ST_Contains(ST_GeomFromText(d.geometryD,4326), ST_SetSRID(ST_Point(u.longitudeU,u.latitudeU),4326))=true AND u.idU = 53;*
Resultado	Nob Hill North Beach Pacific Heights Russian Hill

La segunda consulta (Tabla 28) es idéntica a **C02_1b-3a-4-5** en estructura, sólo que no en los resultados obtenidos. En este caso, se quiere obtener las rutas que pasaron por un determinado distrito, en este caso SOMA. Haciendo dos cambios sobre **C02_1b-3a-4-5** se obtendrán dichas rutas. El primer cambio es modificar la condición de filtrado de la unión, es decir, ahora lo que se quiere es que sólo se escoja el distrito de SOMA en la JOIN, por lo que además de indicar que los puntos deben de estar contenidos dentro de los distritos de *Districts* (mediante

ST_Contains), hay que filtrar los distritos por *nameD* = SOMA. El otro cambio necesario con respecto de **C02_1b-3a-4-5** es en la SELECT, donde se debe seleccionar el identificador de la ruta *IdU*.

Tabla 28: Consulta C03_1b-3a-4-5

ID	**C03_1b-3a-4-5**
Tarjet	***"Rutas que pasaron por el distrito de SOMA"***
Consulta	***SELECT DISTINCT idU*** ***FROM Uber u*** ***JOIN Districts d*** ***WHERE ST_Contains(ST_GeomFromText(d.geometryD,4326),*** ***ST_SetSRID(ST_Point(u.longitudeU,u.latitudeU),4326))=true AND nameD = 'SOMA';***
Resultado	*IdU* de las rutas que cumplan la condición. Un ejemplo de los cinco primeros resultados y los cinco últimos: 4 8 13 15 16 - - - 24987 24993 24996 24998 24999

Sobre tablas PointsInterest/Districts

En este bloque se sigue la misma tónica que en las consultas sobre *Uber* y *Districts*, ya que se trabaja con uniones entre tablas con puntos y distritos. Van a mostrarse distintos casos de búsqueda, y múltiples variantes de los mismos.

En la primera búsqueda de ejemplo (Tabla 29) se busca el nombre de aquellos distritos que tengan más de cuatro puntos de interés dentro de su perímetro. Como puede verse, de nuevo se evalúa ST_Contains en la cláusula WHERE (usando ST_Point para calcular puntos y ST_GeomFromText para calcular polígonos, forzando a estándar WGS84 cada uno) para filtrar los resultados según los puntos de interés de *PointsInterest* estén dentro de los distritos de *Districts* o no. Sobre dicho filtrado, se hace una agrupación con GROUP BY mediante el nombre del distrito, y sobre esta agrupación sólo quedan elegidos aquellos distritos que posean más de cuatro puntos de interés. Para este último filtro se utiliza la cláusula HAVING, la cual se puede utilizar junto a la cláusula GROUP BY para filtrar resultados directamente de las agrupaciones. Al filtrado de HAVING se le indica como condición que la función de agregación

COUNT, aplicada sobre el nombre del punto de interés de la ficha de usuario, devuelva más de cuatro puntos de interés.

Tabla 29: Consulta C04_1b-3a-4-5

ID	**C04_1b-3a-4-5**
Tarjet	***"Distritos con más de cuatro puntos de interés"***
Consulta	***SELECT nameD*** ***FROM districts d*** ***JOIN pointsinterest p*** ***WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326),*** ***ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true*** ***GROUP BY nameD HAVING COUNT (namePI) > 4;***
Resultado	Financial District Fishermans Wharf Golden Gate Park Potrero Hill

Se pueden obtener múltiples variantes a partir de la consulta **C04_1b-3a-4-5**. Una de ellas (Tabla 30) consiste en buscar los distritos que tengan parques (ampliable para cualquier otro tipo de punto de interés). Para ello, se parte de la consulta **C04_1b-3a-4-5**. Si se deja la agrupación de GROUP BY sin el filtro de HAVING, se consigue un listado distintivo (idéntico a aplicar DISTINCT en la SELECT) de los nombres de los distritos. Como sólo se necesitan los nombres de los distritos que tienen parques, se añade con la directiva AND un filtrado por el *typePI* para que sólo queden seleccionados nombres de distritos con parques en su perímetro.

Tabla 30: Consulta C05_1b-3a-4-5

ID	**C05_1b-3a-4-5**
Tarjet	***"Distritos que tienen parques"***
Consulta	***SELECT nameD*** ***FROM districts d*** ***JOIN pointsinterest p*** ***WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326),*** ***ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true AND typePI = 'parque'*** ***GROUP BY nameD;***
Resultado	Financial District Fishermans Wharf Golden Gate Park Potrero Hill

Más búsquedas de utilidad. Se supone el caso en que se quiera conocer la cifra de distritos que tienen un determinado punto de interés, por ejemplo, hoteles (Tabla 31). Como ya se ha visto en los casos de ejemplo anteriores de esta rama, se debe hacer una unión de *Districts* y *PointsInterest* usando la función ST_Contains, evaluando si cada punto está incluido en cada distrito. Partiendo de esta base, como sólo va a interesar contar distritos cuando contengan

hoteles, al filtro WHERE de la unión espacial de ST_Contains se añade otro con la directiva AND sobre el atributo de *PointsInterest typePI*, forzando que el tipo de localización sean hoteles (misma operación que se hacía en la consulta **C05_1b-3a-4-5**). A diferencia que en **C05_1b-3a-4-5**, no se hace agrupación con GROUP BY, no es necesario para esta operación, pero cabe decir que se obtendría idéntico resultado de haberlo usado. Por último, para contar los distritos, se selecciona en la SELECT el resultado de aplicar la función de agregación COUNT sobre el nombre del distrito, *nameD*.

Tabla 31: Consulta C06_1b-3a-4-5

ID	**C06_1b-3a-4-5**
Tarjet	***"Número de distritos que tienen hoteles"***
Consulta	***SELECT COUNT(nameD) FROM districts d JOIN pointsinterest p WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326), ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true AND p.typePI = 'hotel';***
Resultado	Bernal Heights Chinatown Fishermans Wharf Golden Gate Park Haight Marina Mission District Noe Valley North Beach Potrero Hill Russian Hill Sunset District The Castro Western Addittion

A continuación se muestra otro ejemplo aplicando el conteo sobre toda la fila, COUNT (*) (Tabla 32). Donde se quiere averiguar el caso contrario a **C06_1b-3a-4-5**, es decir, el número de parques que tiene un determinado distrito (distrito de Marina en este ejemplo). Conseguir esta variante es sencillo, sólo hay que aplicar dos cambios sobre **C06_1b-3a-4-5**. El primero de ellos consiste en añadir un filtro adicional en la WHERE para seleccionar del resultado de la unión espacial sólo las filas cuyo distrito *nameD* sea Marina, además del filtro ya conocido en **C06_1b-3a-4-5** que filtraba por parques en el atributo *typePI*. El segundo cambio es sobre la función de agregación COUNT de la SELECT, la cual en este caso se aplica sobre la totalidad de la tupla resultante de la unión mediante JOIN. Se podría obtener idéntico resultado aplicando este conteo sobre el punto generado a partir de las coordenadas del punto de interés: COUNT (ST_Point (*longitudePI, latitudePI*)).

Tabla 32: Consulta C07_1b-3a-4-5

ID	**C07_1b-3a-4-5**
Tarjet	***"Número de parques que tiene el distrito de Marina"***

Consulta	*SELECT COUNT(*) FROM districts d* *JOIN pointsinterest p* *WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326),* *ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true* *AND d.nameD = 'Marina'* *AND p.typePI = 'parque';*
Resultado	2

Como último ejemplo para uniones entre puntos de interés y distritos de la rama **1b-3a-4-5**, se ha elegido una *query* más compleja (Tabla 33). Se busca seleccionar el nombre y tipo de todos los puntos de interés que tenga en su perímetro el distrito más grande de San Francisco. Esta unión ya no consiste únicamente en hacer JOIN sobre las dos tablas y aplicar filtrado, si no que requiere trabajar una de las partes implicadas, la tabla *Districts*. La consulta une espacialmente (tal como se ha visto en esta rama, usando ST_Contains) todos los registros de *PointsInterest* con un equivalente de la consulta **C10_1a-2a-4-5**, en la cual se calculaba el distrito más grande de San Francisco. La única diferencia es la no inclusión de los metadatos espaciales en la función ST_GeomFromText, pues no es necesario, ya que no se trabaja con funciones espaciales que hagan uso de ellos. A modo de recordatorio, la consulta **C10_1a-2a-4-5** usaba la función ST_Area sobre la geometría que calculaba ST_GeomFromText, y se ordenaban las filas con ORDER BY evaluando las áreas calculadas.

Tabla 33: Consulta C08_1b-3a-4-5

ID	**C08_1b-3a-4-5**
Tarjet	***"Puntos de interés y su tipo que tiene el distrito más grande de San Francisco"***
Consulta	*SELECT namePI, typePI* *FROM (* *SELECT d.geometryD geometria , ST_Area(ST_GeomFromText(d.geometryD))area* *FROM districts d ORDER BY area DESC LIMIT 1)distrito* *JOIN (* *SELECT * FROM pointsinterest p) punto* *WHERE* *ST_Contains(ST_GeomFromText(distrito.geometria,4326),ST_SetSRID(ST_Point(punto.longitudePI,punto.latitudePI),4326)) = true;*
Resultado	Abraham Lincoln High School centro de estudios Sunset Reservoir parque Sunset Parkway parquet Tonight Soju Bar pub

Sobre tablas Uber/PointsInterest/Districts

Para ampliar más los ejemplos de uniones espaciales de varias tablas usando cláusulas, se van a mostrar dos casos de consultas sobre las tres tablas que componen el caso práctico de este trabajo. En ambas se utilizan cláusulas de filtrado, y en la segunda además se verá un caso completo de uso de filtrado agrupación y ordenación. La estructura mostrada es una manera de hacer unión de tres tablas usando funciones espaciales en los filtrados.

La primera consulta de ejemplo (Tabla 34) busca los puntos de interés que poseen aquellos distritos por los que pasó una determinada ruta, aquella que tiene *IdU* = 5780. Se devuelve el nombre del punto de interés *namePI* y el nombre del distrito *nameD* de los seleccionados. Se puede ver que la JOIN está construida como una secuencia de uniones seguidas: Tabla JOIN Tabla JOIN Tabla. Sobre esta unión triple de todo con todo, se indica mediante filtros WHERE los criterios de unión. En este caso, existen tres condiciones para seleccionar registros de la tabla resultante: Primero, se evalúa que los puntos de las rutas *Uber* estén dentro de los distritos de *Districts* mediante el uso de la función ST_Contains (forzando las tres geografías en estándar WGS84 con id 4326); segundo, se realiza exactamente la misma evaluación sobre los puntos de interés de *PointsInterest* y los distritos de *Districts*; y tercera y última comprobación, se fuerza a que los puntos que se seleccionen de *Uber* sean los de la ruta con *IdU* = 5780. Los tres filtros se unen mediante directivas AND. Sobre esta sub consulta, se aplica una SELECT en la consulta genérica, la cual selecciona el nombre *namePI* del punto de interés y el nombre *nameD* del distrito.

Tabla 34: Consulta C09_1b-3a-4-5

ID	**C09_1b-3a-4-5**
Tarjet	***"Puntos de interés de los distritos por donde pasó la ruta 5780"***
Consulta	***SELECT DISTINCT total.namePI, total.nameD FROM(SELECT * FROM districts d JOIN pointsinterest p JOIN uber u WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326), ST_SetSRID(ST_Point(u.LongitudeU, u.LatitudeU),4326))= true AND ST_Contains(ST_GeomFromText(d.GeometryD,4326), ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true AND u.IDU = 5780)total;***
Resultado	Cable Car Museum Nob Hill California Pacific Medical Center Pacific Heights Calvary Presbyterian Church Pacific Heights Chinatown Merchantas Association Chinatown Far East Café Chinatown German Consulate General Pacific Heights Grace Cathedral Nob Hill The Fairmont San Francisco Nob Hill Willie Woo Woo Wong Playground Chinatown

La segunda consulta de ejemplo sobre esta rama (Tabla 35) es muy completa, ya que se realiza usando los tres tipos de cláusulas posibles en *Hive*. La consulta devuelve el número de rutas que pasaron por distritos que tienen edificios religiosos en su perímetro. Esta consulta es una combinación de varios ejemplos que se han visto en otras ramas más simples. Como en el ejemplo **C08_1b-3a-4-5**, en esta *query* se realiza una triple unión mediante el uso de la cláusula JOIN sin hacer uso de sub consultas. El criterio de unión se establece una vez más con la cláusula de filtrado WHERE, en la cual se indica que tanto los puntos de interés de *PointsInterest* como los puntos de las rutas de *Uber* deben de estar contenidos en los distritos de

Districts (uso de ST_Contains entre distritos y puntos de ambas tablas). Como sólo interesan los distritos con edificios religiosos en su haber, se fuerza a que el *typePI* sea igual a 'edifico religioso'. Para realizar el conteo se necesita agrupar los resultados por el *IdU* mediante GROUP BY, y seleccionar el número de fila con la cláusula SELECT. Para que la selección sea correcta, se ordenan los resultados por el número de fila 'fila', seleccionado en la SELECT, e indicar mediante la directiva LIMIT que sólo quede seleccionado el primer resultado, el cual se corresponderá con el número de filas del conjunto resultado de la consulta (y que en este caso, por el uso de GROUP BY es igual al número de rutas).

Tabla 35: Consulta C11_1b-3a-4-5

ID	**C11_1b-3a-4-5**
Tarjet	***"Número de rutas que pasaron por distritos que tienen edificios religiosos"***
Consulta	***SELECT ROW_NUMBER( ) OVER( ) fila*** ***FROM districts d*** ***JOIN pointsinterest p*** ***JOIN uber u*** ***WHERE ST_Contains(ST_GeomFromText(d.GeometryD,4326),*** ***ST_SetSRID(ST_Point(u.LongitudeU, u.LatitudeU),4326))= true*** ***AND ST_Contains(ST_GeomFromText(d.GeometryD,4326),*** ***ST_SetSRID(ST_Point(p.LongitudePI, p.LatitudePI),4326))=true*** ***AND typePI='edificio religioso' GROUP BY idU ORDER BY fila DESC LIMIT 1;***
Resultado	7050

3.6.6. Varias tablas – Unión no espacial – Sin cláusulas (1b-3b-4a-5)

Esta rama recoge aquellas consultas entre varias tablas que no contienen ningún tipo de cláusula, ni para filtrar los resultados ni para efectuar la combinación de tablas. Este tipo de búsquedas no suelen utilizarse, primero por lo ineficientes que pueden llegar a ser si el número de datos a combinar es alto; y segundo, no tiene prácticamente utilidad no usar ninguna cláusula, ya se de filtrado, ordenación o agrupación. Por ello, en este punto se muestra una única consulta de ejemplo, ya que aunque este tipo de búsqueda apenas se usen son válidas de igual forma.

El esquema canónico para estas consultas puede verse en la Ilustración 18. Sólo cabe el uso de funciones de agregación o espaciales a la hora de seleccionar resultados, no existiendo posibilidad de realizar ninguna operación que conlleve usar cláusulas adicionales.

```
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
JOIN
SELECT B1|Function (B1), B2|Fun (B2)… Bn|Function (Bn)
FROM TablaB
[JOIN
SELECT N1|Function (N1), N2|Fun (N2)… Nn|Function (Nn)
FROM TablaA ]
```

Ilustración 18: Esquema canónico para la rama 1b-3b-4a-5

Sobre tablas PointsInterest/Districts

La consulta que se propone de ejemplo (Tabla 36) no adquiere sentido en un ámbito real de implementación, sirve sólo a modo de ejemplo. Con la consulta se devuelve el número de combinaciones entre distritos de *Districts* y puntos de interés de *PointsInterest*. Se ha realizado aplicando la función COUNT sobre el punto generado sobre el punto de interés. Como ya se ha dicho, sólo es necesario saber que existe este tipo de posibilidad, pero no tiene uso alguno.

Tabla 36: Consulta C01_1b-3b-4a-5

ID	**C01_1b-3b-4a-5**
Tarjet	***"Número de combinaciones totales de Distritos – Puntos de interés"***
Consulta	***SELECT COUNT(ST_Point(PointsInterest.longitudePI,PointsInterest.latitudePI)) FROM PointsInterest JOIN Districts;***
Resultado	1420

3.6.7. Varias tablas – Unión no espacial – Con cláusulas (1b-3b-4b-5)

Esta rama describe aquellas consultas que actúan sobre varias tablas distintas, donde la combinación entre ellas no se realiza a través de sus datos espaciales (como en la rama **1b-3a-4-5**) si no a través de otros atributos. Además pueden utilizarse cláusulas para filtrar los resultados, agruparlos u ordenarlos según convenga.

El esquema general de implementación puede consultarse en la Imagen 19. Es idéntico al de la rama **1b-3a-4-5**, sólo que con la diferencia de que no se utiliza funciones espaciales en los filtros ON o WHERE para establecer condiciones de unión.

```
SELECT A1|Function (A1), A2|Fun (A2)… An|Function (An)
FROM TablaA
[WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaA.A2|
Function (TablaA.A2)]… [AND|OR TablaA.An| Function (TablaA.An)]]
[GROUP BY TablaA.atribute/s|Fun (TablaA.atribute/s)
[HAVING COUNT (TablaA.atribute) (>|=) value)]]
[ORDER BY TablaA.atribute|Function (TablaA.atribute)]
JOIN
SELECT B1|Function (B1), B2|Fun (B2)… Bn|Function (Bn)
FROM TablaB
[WHERE TablaB.B1|Function (TablaB.A1) [AND|OR TablaB.B2|
Function (TablaB.B2)]… [AND|OR TablaB.Bn| Function (TablaB.Bn)]]
[GROUP BY TablaB.atribute/s|Fun (TablaB.atribute/s)
[HAVING COUNT (TablaB.atribute) (>|=) value)]]
[ORDER BY TablaB.atribute|Function (TablaB.atribute)]
 [JOIN
SELECT N1|Function (N1), N2|Fun (N2)… Nn|Function (Nn)
FROM TablaA
[WHERE TablaN.N1|Function (TablaN.N1) [AND|OR TablaN.N2|
Function (TablaN.N2)]… [AND|OR TablaN.Nn| Function (TablaN.Nn)]]
[GROUP BY TablaN.atribute/s|Fun (TablaN.atribute/s)
[HAVING COUNT (TablaN.atribute) (>|=) value)]]
[ORDER BY TablaN.atribute|Function (TablaN.atribute)] n veces
ON|WHERE TablaA.A1|Function (TablaA.A1) [AND|OR TablaB.B2|
Function (TablaB.B2)]… [AND|OR TablaN.Nn| Function (TablaN.Nn)]
```

Ilustración 19: Esquema canónico para la rama 1b-3b-4b-5

Sobre tablas Uber/PointsInterest

Se proponen tres ejemplos que ejecutan sobre las tablas de *Uber* y *PointsInterest*. Como en otras ramas de esta memoria, los ejemplos proponen casos parecidos de búsqueda para resaltar las distintas maneras que existen de conseguir resultados idénticos o parecidos. Los tres ejemplos giran en torno a conseguir encontrar la ruta *Uber* que, o bien, pasó más cerca de Fort Point, o bien la que pasó más lejos de ese punto.

El primer ejemplo propuesto (Tabla 37) trata de encontrar la ruta *Uber* (*IdU* y fecha y hora de registro del punto) que pasó más cerca de Fort Point. En este caso, la combinación se realiza filtrando el total de filas por el nombre del punto de interés, el cual recibe el alias de *nombre*, igualándolo al valor "Fort Point". Por otro lado, la SELECT de la sub consulta más interna calcula la distancia entre cada punto mediante ST_Distance (los puntos dados en estándar WGS84 con ST_SetSRID). La siguiente sub consulta inmediatamente superior selecciona esa distancia para usarse en el ORDER BY, y la ordenación se realiza en sentido ascendente con ASC y se limitan los resultados a 1, para quedarnos con el punto más cercano. Finalmente, de ese punto se extrae el *IdU* y su fecha y hora de registro con la aplicación de *Uber*.

Tabla 37: Consulta C01_1b-3b-4b-5

ID	**C01_1b-3b-4b-5**
Tarjet	***"Id, tiempo y fecha de la ruta Uber que pasó más cerca de Fort Point"***
Consulta	***SELECT id, dt FROM(SELECT total.id, total.dt,total.distancia FROM(SELECT uber.idU id,uber.datetimeU dt,pointsinterest.namePI nombre,ST_Distance(ST_SetSRID(ST_Point(uber.longitudeU, uber.latitudeU),4326),ST_SetSRID(St_Point(Pointsinterest.longitudePI,Pointsinterest.latitudePI),4326)) distancia FROM uber JOIN Pointsinterest)total WHERE total.nombre = 'Fort Point' ORDER BY distancia ASC LIMIT 1)total2;***
Resultado	22309 2007-01-05 19:08:09+00:00

La segunda consulta de ejemplo es idéntica a la anterior en estructura y forma, sólo que ahora se busca la ruta *Uber* que pasó más lejos de Fort Point (Tabla 38). Para ello el único cambio a efectuar es modificar la instrucción ASC por DESC.

Tabla 38: Consulta C02_1b-3b-4b-5

ID	**C02_1b-3b-4b-5**
Tarjet	***"Id, tiempo y fecha de la ruta Uber que pasó más lejos de Fort Point (unión con WHERE)"***
Consulta	***SELECT id, dt FROM(SELECT total.id, total.dt,total.distancia FROM(SELECT uber.idU id,uber.datetimeU dt,pointsinterest.namePI nombre,ST_Distance(ST_SetSRID(ST_Point(uber.longitudeU, uber.latitudeU),4326),ST_SetSRID(St_Point(Pointsinterest.longitudePI,Pointsinterest.latitudePI),4326)) distancia FROM uber***

	JOIN Pointsinterest)total *WHERE total.nombre = 'Fort Point'* *ORDER BY distancia DESC LIMIT 1)total2;*
Resultado	21578 2007-01-02 03:43:21+00:00

La última consulta de ejemplo (Tabla 39) es una variante de la consulta anterior **C03_1b-3b-4b-5**. La variación consiste en realizar la combinación entre tablas mediante el uso de la cláusula ON. Para ello, se debe sustituir la cláusula WHERE por ON, donde la condición consiste en igualar el *namePI* de *PointsInterest* = "Fort Point".

Tabla 39: Consulta C03_1b-3b-4b-5

ID	**C03_1b-3b-4b-5**
Tarjet	***"Id, tiempo y fecha de la ruta Uber que pasó más lejos de Fort Point (unión con ON)"***
Consulta	*SELECT id, dt FROM(* *SELECT total.id, total.dt,total.distancia FROM(* *SELECT uber.idU id,uber.datetimeU dt,pointsinterest.namePI nombre,ST_Distance(ST_SetSRID(ST_Point(uber.longitudeU, uber.latitudeU),4326),ST_SetSRID(St_Point(Pointsinterest.longitudePI,Pointsinterest.latitudePI),4326)) distancia* *FROM uber* *JOIN Pointsinterest ON Pointsinterest.namePI = 'Fort Point')total* *ORDER BY distancia DESC LIMIT 1)total2;*
Resultado	21578 2007-01-02 03:43:21+00:00

3.7. Pruebas de consultas espaciales

Este punto describe el proceso de prueba de alguna de las consultas más significativas de las detalladas en *3.6. Generación asistida de consultas espaciales.* El objetivo es comprobar el correcto funcionamiento de algunas de las consultas implementadas. Según el tipo de consulta y los datos que maneje se han diseñado métodos de prueba distintos, mediante reconversión de consultas, el uso de medios visuales (QGIS o Google Maps) o pruebas de caja negra.

Se han seleccionado cuatro consultas para probar. Son las siguientes: **C01_1a-2a-4b-5, C08_1a-2a-4b-5, C03_1a-2b-3b-4-5** y **C01_1b-3b-4b-5**.

3.7.1. Prueba consulta C01_1a-2a-4b-5

Esta consulta pretende dar certeza de que todos los registros contenidos en la tabla base de trabajo *Uber*, contienen un punto geográfico válido. Resulta de mucha utilidad esta prueba, ya que se verifica que se está trabajando sobre un *data set* válido, además de proporcionar el número total de registros que no es capaz de dar Microsoft Excel. Se realizará un total de dos *querys* sobre *Uber*. La primera consulta contiene la función espacial, y la segunda sin ella. Ambas deben de devolver como resultado una única fila con un único atributo, la cual debe ser la misma cifra numérica correspondiente al número de filas de las tablas.

La prueba para esta consulta es trivial y muy sencilla (Tabla 40). Para demostrar que **C01_1a-2a-4b-5** funciona correctamente se comparará que el valor devuelto por COUNT (*) de la consulta espacial sea igual que el devuelto por búsqueda a la totalidad de las filas.

Tabla 40: Consulta de prueba C01_1a-2a-4b-5

Consulta prueba C01_1a-2a-4b-5
SELECT count(*) ***FROM Uber;***

Esta consulta devuelve como resultado el número total de filas de la tabla. Al tener un número de registros muy alto en el caso de *Uber* se debe hacer uso de esta técnica para el cálculo de filas.

Para *Uber* ejecutando la consulta **C01_1a-2a-4b-5** se obtienen 1128663 registros, tal como se puede ver en la Ilustración 20.

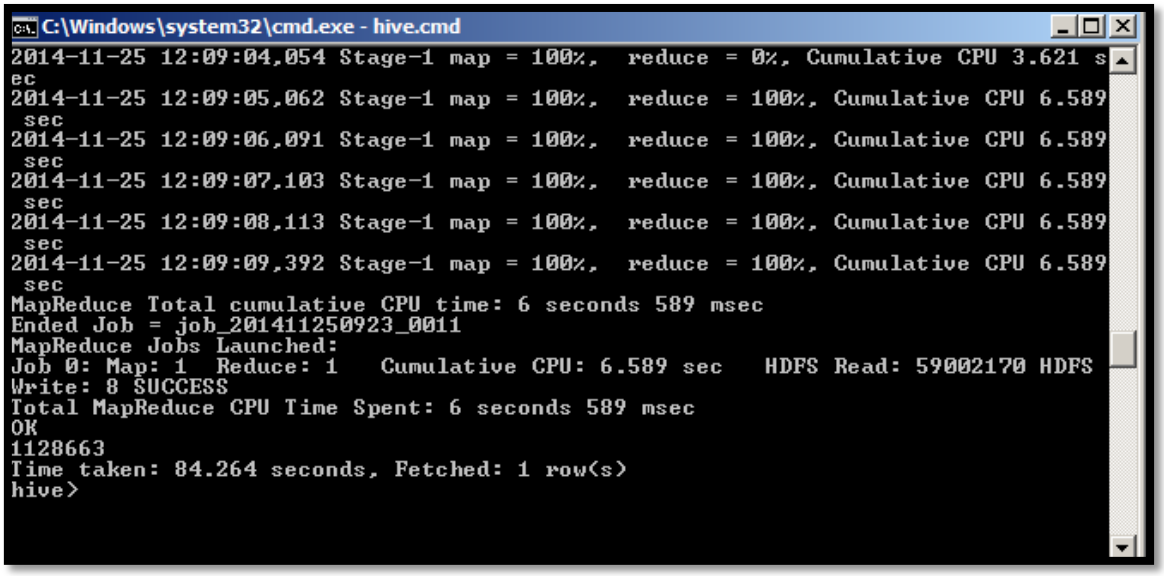

Ilustración 20: Consulta 1: Ejecución de C01_1a-2a-4b-5 sobre *Uber*

Cuando se ejecuta la consulta de la Tabla 40 sobre *Uber* se obtienen exactamente el mismo número de registros que con la consulta **C01_1a-2a-4b-5**, 1128663 filas. Puede verse este resultado en la Ilustración 21.

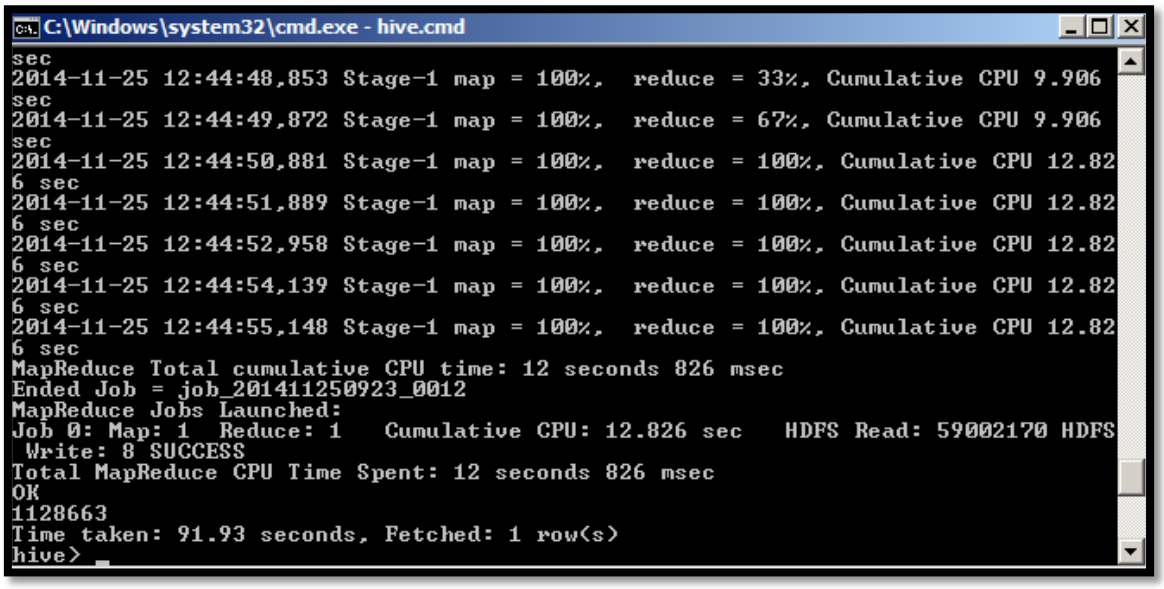

Ilustración 21: Consulta 1: Ejecución de consulta prueba de C01_1a-2a-4b-5

Atendiendo a los resultados, se demuestra que todas las filas de la tabla *Uber* guardan una localización geográfica en forma de punto.

3.7.2. Prueba consulta C10_1a-2a-4-5

Esta consulta obtiene el nombre del distrito más grande del San Francisco: Sunset District. La consulta se ejecuta sobre la tabla *Districts*. La prueba tiene como objetivo conocer cómo utilizar una de las funciones más comunes de PostGIS: el cálculo de áreas en polígonos. Por otro lado, el ejercicio con áreas también sirve para reflejar la carencia de algunas funciones en Hive para manejar unidades geográficas.

Si se ejecuta **C10_1a-2a-4-5** en Hive se obtiene como resultado el distrito de Sunset District. Se puede ver este resultado en la Ilustración 22.

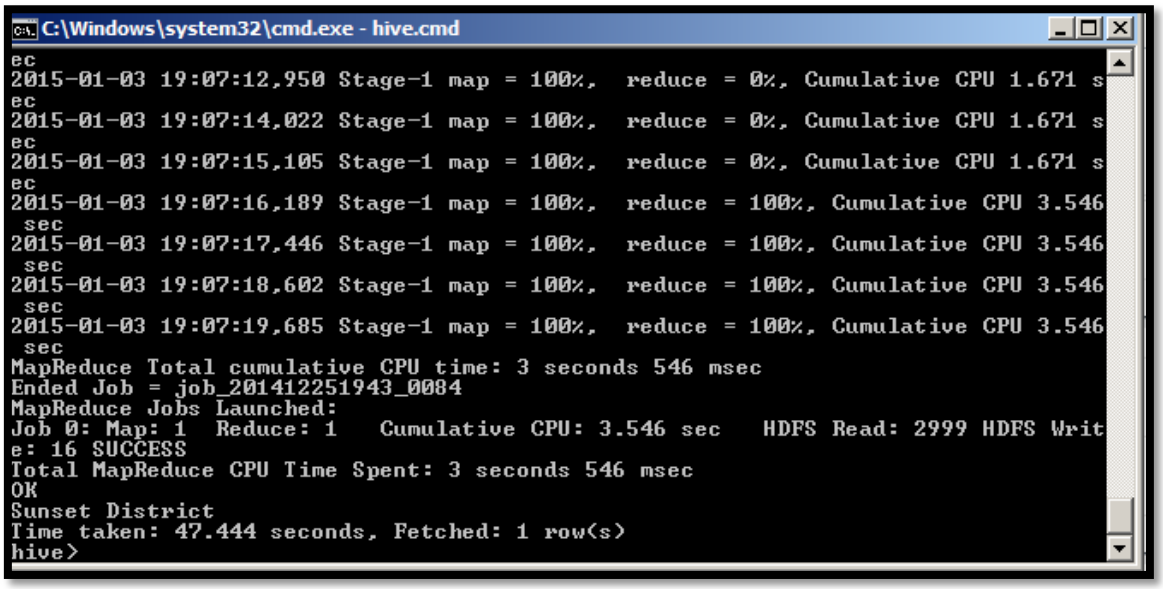

Ilustración 22: Ejecución de C10_1a-2a-4-5 sobre *Districts*

La función ST_Area proporciona en este caso los cálculos en unidades geométricas del SRID que se esté utilizando. Este valor sirve de indicativo para medir tamaños, pero no son unidades conocidas para medir áreas. Al no haber en Hive una función que transforme estas unidades en, por ejemplo, metros cuadrados, no se dispone de unidades legibles. No obstante, sí sirve para hacer cálculos orientativos, puesto que se ha podido demostrar en este caso práctico, como se verá a continuación, que a mayor valor de ST_Area, mayor es el área del distrito.

Se va a proceder a la demostración de que la función cumple su objetivo de manera correcta. Para ello, se ejecutará la consulta **C10_1a-2a-4-5** con la cláusula LIMIT fijada en 5 para obtener los distritos más grandes además del primero, y se seleccionará en la SELECT general además del nombre del distrito el área calculada para cada uno. Con estos cambios se verá que la función cumple con los cálculos. Se compararán los valores de la lista de distritos obtenidos en Hive con los valores que tienen los distritos en metros cuadrados, observando si la clasificación obtenida es correcta. La nueva consulta con estos cambios aplicados para realizar la prueba es la indicada en la Tabla 41.

Tabla 41: Consulta de prueba C10_1a-2a-4-5

Consulta C10_1a-2a-4-5 (Con LIMIT 5 y SELECT general con Area)
SELECT b.nameD, b.area ***FROM (SELECT d.nameD, ST_Area(ST_GeomFromText(d.geometryD,4326)) area*** ***FROM Districts d*** ***ORDER BY area DESC*** ***LIMIT 5) b ;***

Al ejecutar la consulta en Hive se obtiene la salida que se esperada, tal como se puede ver en la Ilustración 23. Se obtienen los cinco distritos más grandes de San Francisco y su área proporcionada en unidades espaciales de ST_Area. Como se puede ver, las cifras de áreas que devuelve la función son pequeñas, debido a que el área se calcula con la geometría calculada a

partir de las longitudes y latitudes de los distritos, las cuales están expresadas con mucha precisión. En el caso de las geometrías que se manejan en este caso práctico para los distritos, el cálculo que realiza ST_Area es sobre un polígono de cuatro lados (base por altura).

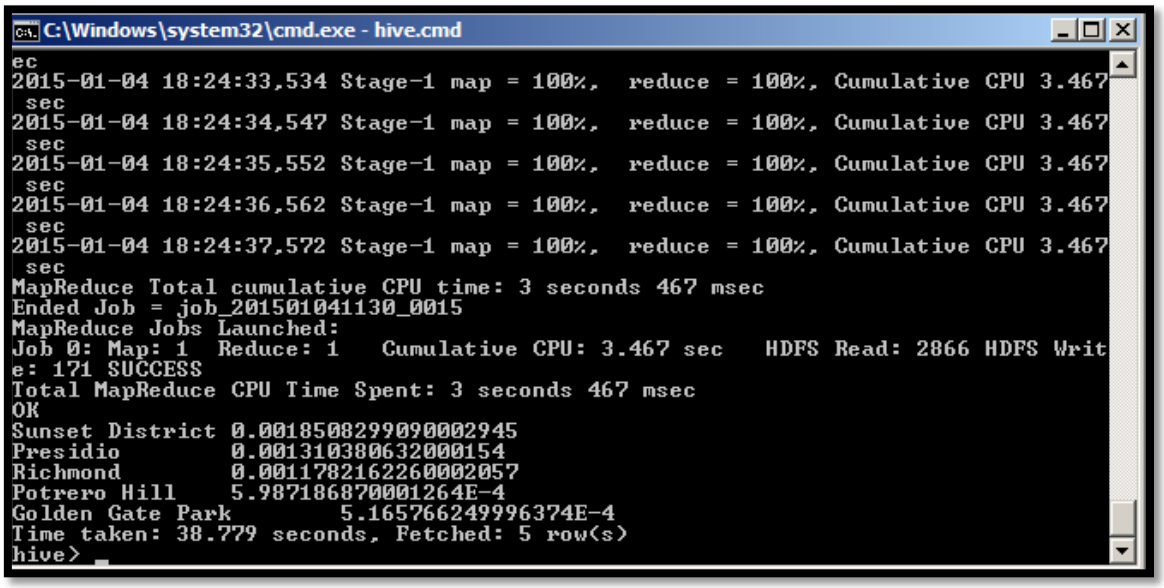

Ilustración 23: Ejecución de consulta prueba de C10_1a-2a-4-5

Finalmente, se muestra a continuación en la Tabla 42 una comparativa de las áreas de los cinco distritos expresadas en metros cuadrados y en unidades espaciales de ST_Area, donde se puede ver que los cálculos de esta función son correctos. Los distritos aparecen ordenados de mayor a menos área. Para obtener las unidades en metros cuadrados se ha usado Google Maps para mapear los distritos con sus longitudes y latitudes y se ha observado el área resultante. En el punto *3.4. Creación de base de datos* se detalló el diseño de *Districts*, donde se explicaba que la creación de los polígonos de los distritos para este caso práctico eran aproximaciones de las áreas de los distritos reales, ya que en realidad éstos no tienen forma de figura de cuatro lados pares dos a dos, sino que en su mayoría son irregulares. Por esta razón, las áreas de los distritos reales difieren ligeramente de las obtenidas de los polígonos de *Districts*, siendo menores las reales que las obtenidas de las coordenadas de *Districts*.

Tabla 42: Área de los cinco distritos más grandes de san Francisco

	Distrito	Área (Km cuadrados)	Área (unidades ST_Area)
1	Sunset District	18.16	0.0018508299090002945
2	Presidio	12.81	0.0013103806320000154
3	Richmond	11.51	0.0011782162260002057
4	Potrero Hill	5.85	5.987186870001264E-4
5	Golden Gate Park	5.04	5.165766249996374E-4

Si en Hive se contara con una biblioteca que contuviera una función como ST_Transform de PostGres el cálculo de áreas en metros cuadrados sería trivial y no hubiera sido necesario el

mapeo manual para averiguarlo. Es una de las líneas de trabajo futuras que podrían abrirse para aumentar la cobertura de datos espaciales en Hive, comentada en el punto *4. Conclusiones finales. Futuras líneas de trabajo*.

3.7.3. Prueba consulta C03_1a-2b-3b-4-5

Esta prueba tiene como objetivo comprobar el correcto funcionamiento de la función ST_Distance. Para ello se seguirá un procedimiento similar al utilizado para probar C10_1a-2a-4-5. Se construirá una consulta que permita ampliar el horizonte de los resultados obtenidos, y se demostrará que los distritos colindantes que devuelve la *query* son realmente los vecinos de Chinatown.

Al ejecutar **C03_1a-2b-3b-4-5** sobre *Districts* se obtienen los resultados de la Ilustración 24. Chinatown tiene un total de tres distritos pegados a sus límites. Existe un cuarto, Tenderloin, el que no estaba incluido en el fichero *Districts*, el cual limita por el sur con Chinatown. Este distrito no se tiene en cuenta.

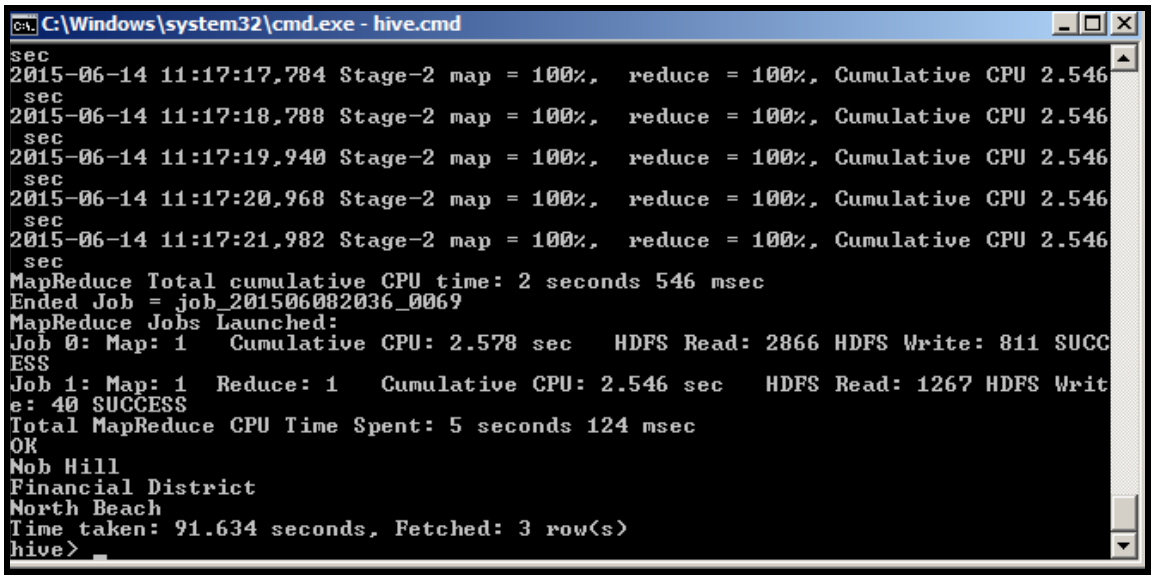

Ilustración 24: Ejecución de C03_1a-2b-3b-4-5 sobre *Districts*

La verificación de estos resultados se ha realizado en dos pasos. El primero de ellos ha consistido en modificar **C03_1a-2b-3b-4-5** para que se muestre, además del nombre del distrito, la distancia entre Chinatown y sus colindantes. En la Tabla 43 puede verse que el único cambio añadido sobre **C03_1a-2b-3b-4-5** ha sido seleccionar el alias de la distancia entre distritos, *distancia*, en la SELECT general.

Tabla 43: Consulta de prueba C03_1a-2b-3b-4-5

Consulta C03_1a-2b-3b-4-5 (Con LIMIT 5 y SELECT general con Area)
SELECT completa.nombre,distancia FROM (***SELECT total.nameD nombre, distancia FROM (*** ***SELECT Districts.nameD,***

```
ST_Distance(ST_GeomFromText(Districts.geometryD,4326),ST_GeomFromText(distrito.geo
metryD,4326)) distancia
FROM Districts
JOIN (
SELECT * FROM Districts
WHERE Districts.nameD = 'Chinatown')distrito)total
WHERE total.nameD != 'Chinatown'
ORDER BY distancia ASC LIMIT 5)completa
WHERE completa.distancia = 0;
```

Los resultados de esta consulta de prueba pueden verse en la Ilustración 25. Para los tres distritos colindantes con Chinatown contenidos en *Districts* se obtiene como distancia SRID cero.

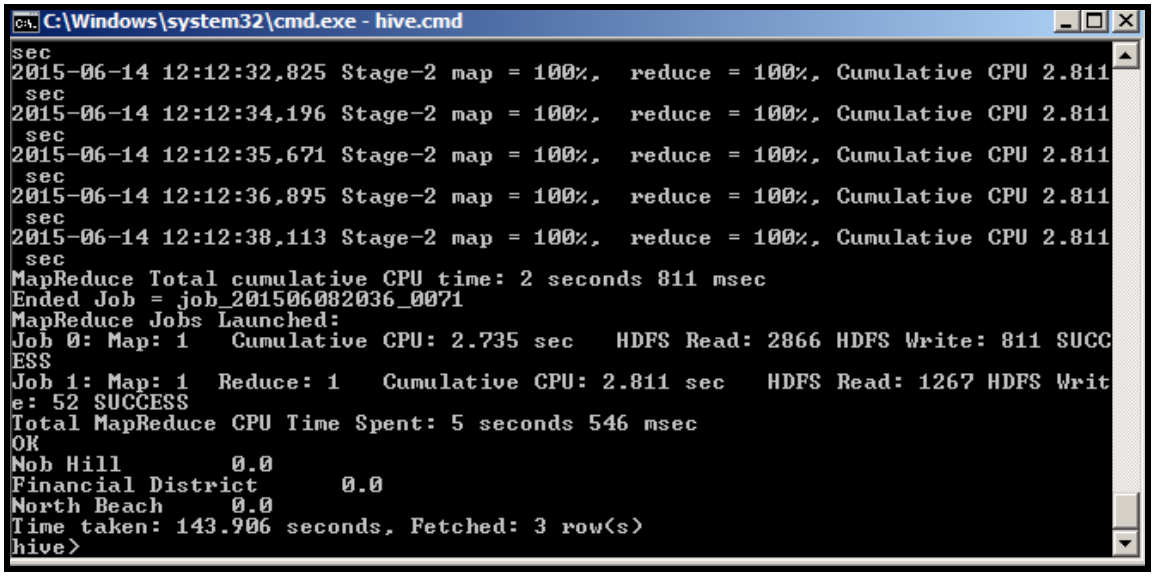

Ilustración 25: Ejecución de consulta prueba de C03_1a-2b-3b-4-5

Con el objetivo de comprobar de forma visual que los resultados son correctos, se ha utilizado Google Maps. En la Ilustración 26 se ha marcado en amarillo los distritos vecinos de Chinatown, y bordeado en rojo el perímetro de Chinatown. La consulta **C03_1a-2b-3b-4-5** funciona correctamente.

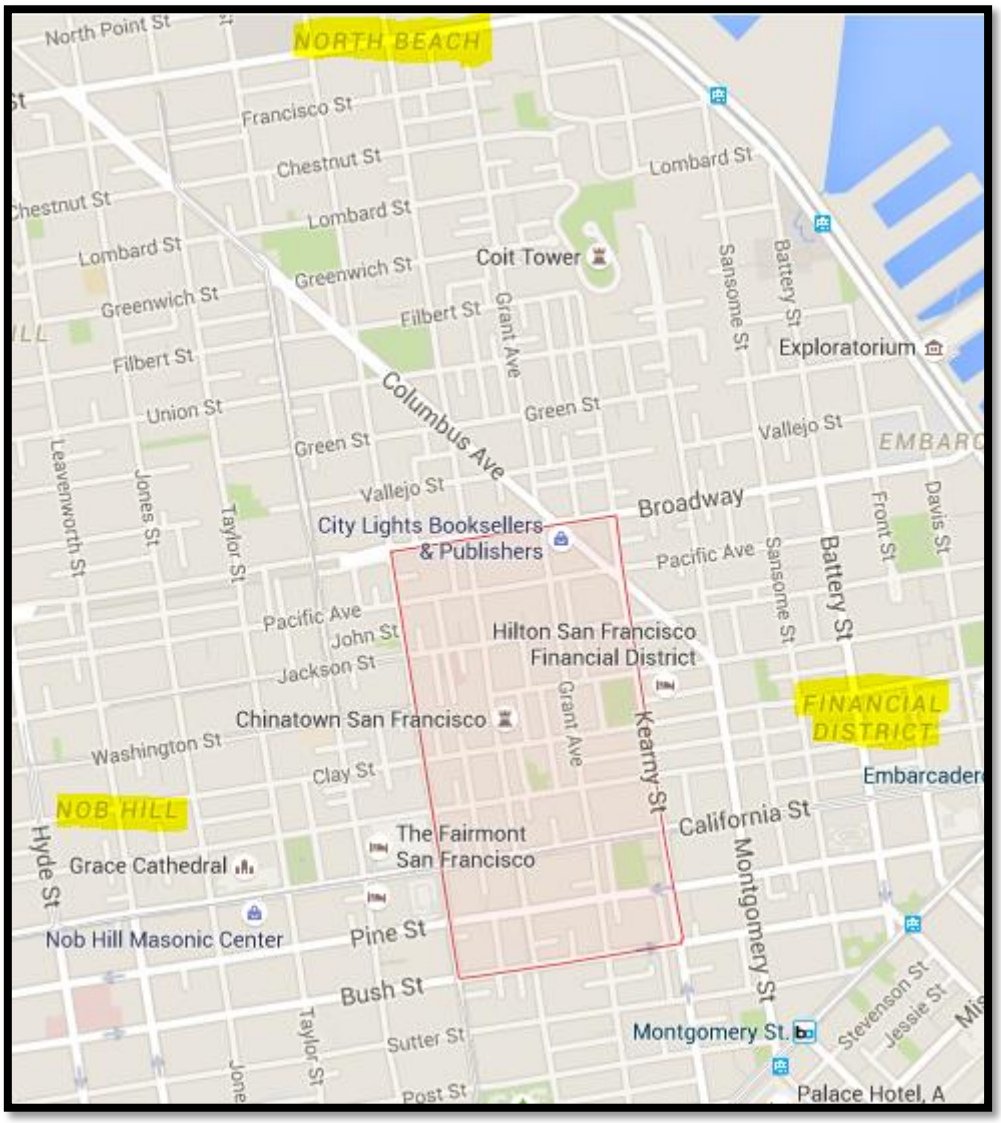

Ilustración 26: Distritos colindantes a Chinatown en Google Maps

El principal problema de la función espacial ST_Distance en Hive es que las unidades que se obtienen al utilizarla no se dan en el Sistema Internacional. Al igual que sucede con ST_Area, en Hive no existe un equivalente a ST_Transform para Hive, con la cual poder obtener geografías, y a partir de esas geografías usar otras funciones sí existentes en Hive con las que calcular medidas en unidades del Sistema Internacional.

3.7.4. Prueba consulta C01_1b-3b-4b-5

La prueba sobre esta consulta busca comprobar que las distancias calculadas con ST_Distance entre puntos de *Uber* Y *PoinsInterest*. La consulta buscaba aquel punto de *Uber* que hubiera pasado más cerca de un punto e interés concreto de *PointsInterest*: Fort Point. Para probar esta *query* una vez más se procederá a hacerlo en dos pasos diferenciados: el primero de ellos consiste en realizar una consulta derivada de **C01_1b-3b-4b-5** que amplíe la información sobre

los datos extraídos; y segundo, mostrar mediante Google Maps la cercanía de los puntos más próximos a Fort Point.

En primer lugar se muestra en la Ilustración 27 el resultado de ejecutar la consulta **C01_1b-3b-4b-5**. Se devuelve una única ruta, la 22309, y la fecha y hora del punto que consta más cercano a Fort Point.

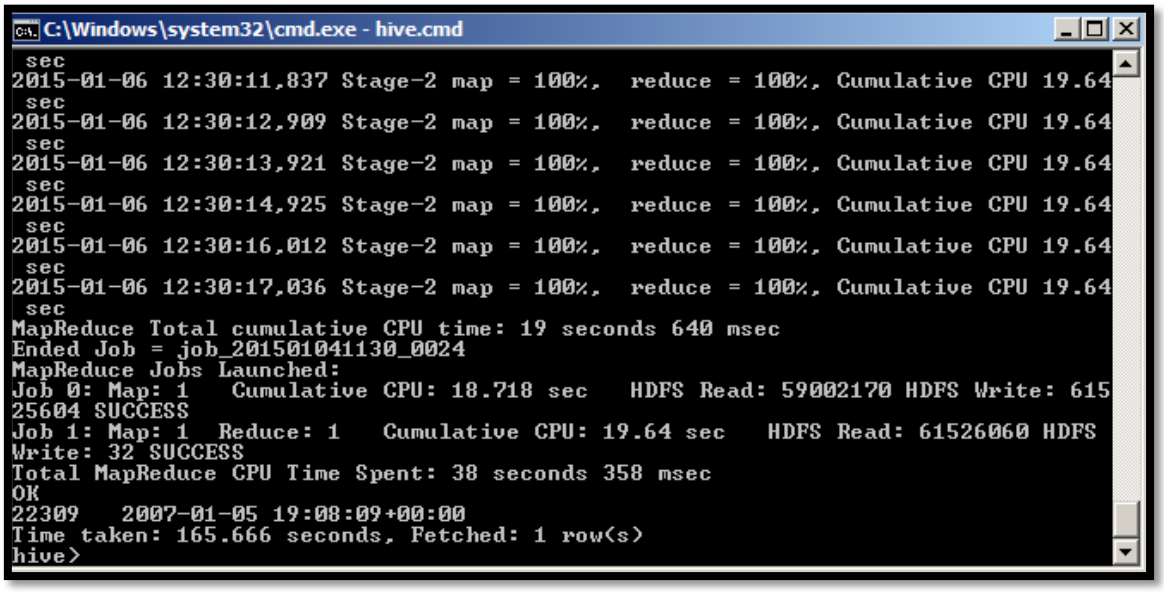

Ilustración 27: Ejecución de C01_1b-3b-4b-5 sobre *Uber* y *PoinstInterest*

Con la información que arroja **C01_1b-3b-4b-5** no es posible determinar muchos detalles acerca de si se ha obtenido la ruta correcta. Por ello, se ha modificado la *query* con el objetivo de obtener más datos acerca de las operaciones que realizado la consulta. En la tabla 44 se muestra la consulta de prueba, en la cual se selecciona también la distancia entre puntos *Uber* y de *PointsInterest* y las coordenadas del punto, y se amplía el número de resultados mostrados, cambiando la instrucción LIMIT 1 por LIMIT 5. Con cinco resultados basta para estudiar el caso.

Tabla 44: Consulta de prueba de C01_1b-3b-4b-5

Consulta C01_1b-3b-4b-5 (Con LIMIT 5)
SELECT id, dt,distancia, longitude, latitud FROM(SELECT total.id, total.dt,total.distancia, longitude, latitud FROM(SELECT uber.idU id,uber.datetimeU dt,pointsinterest.namePI nombre,ST_Distance(ST_SetSRID(ST_Point(uber.longitudeU, uber.latitudeU),4326),ST_SetSRID(St_Point(Pointsinterest.longitudePI,Pointsinterest.latitud ePI),4326)) distancia, uber.longitudeU longitude, uber.latitudeU latitud FROM uber JOIN Pointsinterest)total WHERE total.nombre = 'Fort Point' ORDER BY distancia ASC LIMIT 5)total2;

Con esta consulta, al llevarla a Hive nos devuelve los cinco puntos registrados más cerca de Fort Point, los cuales se corresponden en este caso con cinco rutas distintas. Las distancias vienen dadas en unidades SRID. La Ilustración 28 refleja estos resultados, pudiéndose apreciar que la ruta 22309 fue la que pasó más cerca a este conocido punto de San Francisco.

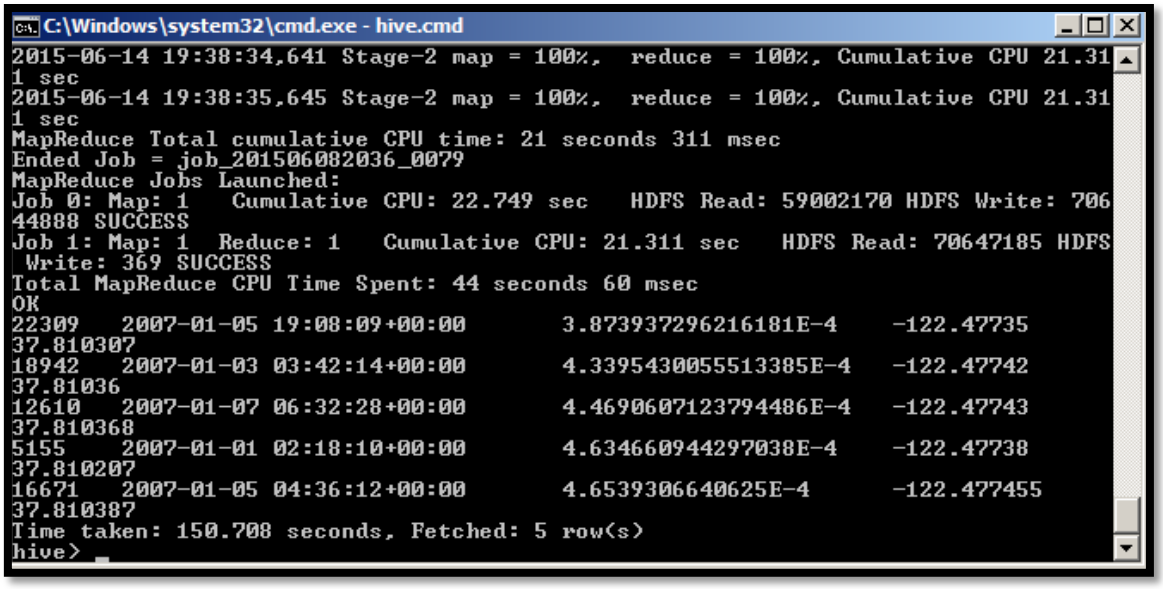

Ilustración 28: Ejecución de consulta prueba de C01_1b-3b-4b-5

Para mostrar de una forma más gráfica los resultados de la consulta, se ha llevado los cinco puntos resultantes de la consulta de prueba a Google Maps, y se han colocado en su localización en Google Maps. En la Ilustración 29 puede verse que todas las rutas que han pasado cerca de Fort Point lo han hecho cruzando el puente Golden Gate, pero en el caso de la ruta 22309 lo ha hecho más pegado al punto de interés.

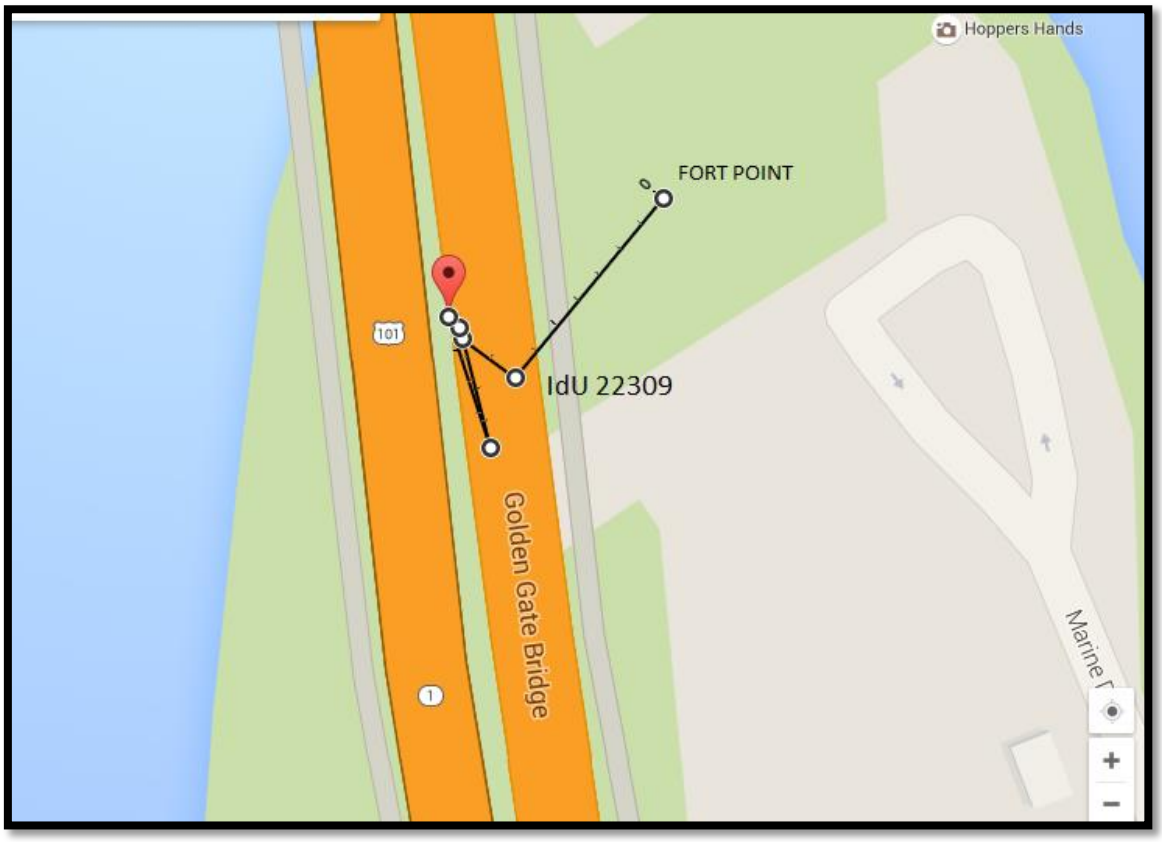

Ilustración 29: Localización de puntos respecto de Fort Point en Google Maps

4. Conclusiones finales. Futuras líneas de trabajo

A la conclusión de este trabajo se puede decir que se han cumplido los dos objetivos principales de este proyecto. Pese a que en un principio dichos objetivos no fueron muy claros debido al enfoque experimental del proyecto, finalmente los que se establecieron han resultado ser, una vez alcanzados, interesantes y útiles para la comunidad de nuevos desarrolladores de Hadoop y en concreto Hive.

En primer lugar, **se ha conseguido elaborar un esquema de implementación de consultas** a partir de los patrones iniciales de los que se partía, gracias a la intensiva experimentación con consultas espaciales realizada sobre entorno de Hadoop. Como bien se describió en el punto *3.6 Generación asistida de consultas espaciales* con este esquema se buscaba dar un uso útil a toda la información y aprendizaje alcanzado con la experimentación. Dicho esquema sirve como introducción al lenguaje de consultas HiveQL. Al haber realizado una experimentación lo suficientemente masiva, puede concluirse que el esquema abarca el diseño de cualquier consulta espacial que se quiera hacer para Hive. Esto hace constatar que los patrones extraídos del Trabajo Fin de Grado *Transformación de las restricciones OCL de un esquema UML a consultas de SQL* del ex alumno Sergio Casillas se pueden aplicar en el contexto de Hive con datos espaciales.

En segundo lugar, el propio documento se ha elaborado pensando en que será utilizado por gente que quiera introducirse en Hive para trabajar con datos espaciales. Por ello, se han incluido todo tipo de guías de configuración en forma de anexos, se han dedicado puntos a explicar paso a paso la creación del caso práctico, se proporcionan múltiples referencias técnicas acerca de dónde encontrar los recursos utilizados y dónde encontrar más información, etc. En definitiva, **el lector encontrará en este documento su mejor aliado para empezar a trabajar con estas tecnologías**.

El trabajo se ha realizado no sin complicaciones. Los baches más importantes encontrados a lo largo del desarrollo del trabajo se resumen en los siguientes:

- **Dificultad para encontrar información técnica muy específica**: Este trabajo ha servido para verificar que el campo en el que se ha desarrollado el proyecto es muy nuevo aún. Resolver determinadas dudas técnicas se convertía en toda una odisea, ya que apenas existían fuentes o comunidades online que traten el tema de datos espaciales en Hive.

- **Ausencia de determinadas bibliotecas espaciales para Hive:** En la web pueden encontrarse prácticamente cualquier versión para Hive de funciones PostGIS. Pero de algunas no hay implementación. Al depender su desarrollo de comunidades independendientes de desarrolladores, como GitHub, no se garantiza tener el equivalente de todas ellas para Hive. Esto ha hecho que no se haya podido avanzar en algunos aspectos del proyecto y tener que tomar otros rumbos. Un ejemplo es el diseño de la geometría de los polígonos en la tabla *Districts* debido al no existir el equivalente de la función ST_Transform de PostGIS para Hive.

- **Depuración y prueba lenta de consultas:** Al trabajar sobre Hadoop/Hive, se hizo con una configuración no distribuida. Hadoop y su *Map Reduce* es más eficiente cuanto más puede dividir la carga de trabajo entre los nodos. Si no se distribuye la carga de trabajo, el coste de ejecución es alto. Esto ha provocado que probar determinadas consultas (sobre todo aquellas que contenían JOIN pesados) haya sido costoso en tiempo.

HQL fue concebido a modo de símil de SQL. Por ello, los parecidos entre uno y otro lenguaje son muchos, llegando al punto en que cualquiera que conozca medianamente bien los fundamentos de cualquier SQL puede empezar de forma sencilla en HQL. No es así de sencillo con respecto al uso de funciones espaciales. Por ello, este trabajo tenía un objetivo no principal, pero no menos importante que los citados a lo largo de la introducción del trabajo. Dicho objetivo ha sido **obtener una serie de diferencias en la sintaxis y acciones que se permite en HQL y las que se permite en un SQL más estándar como PostGres**. Se planteó la idea de implementar un prototipo en lenguaje Java que tradujera consultas en SQL de PostGres a HQL de Hive a partir de ese estudio de diferencias. Finalmente se descartó para centrar la atención en los objetivos básicos del TFG, e incluirlo como futuro proyecto dentro del departamento. Es una de las vías más interesantes que se abren a partir de este trabajo.

Concluir también que no todo el trabajo realizado sobre este proyecto ha podido quedar reflejado en esta memoria. Muchas de las horas que se han dedicado a la investigación y a la experimentación, sobretodo, pueden expresarse en la planificación del proyecto, pero no en el resto del documento. La memoria recoge resultados y datos objetivos que se ha conseguido obtener, pero no deja constancia de los tiempos perdidos en vano intentando encontrar soluciones diversas a problemas a priori difíciles. Una de las dificultades que se pueden poner como ejemplo en este punto ha sido lo novedoso que resulta el medio sobre el que se ha trabajado, lo cual hacía que fuera difícil encontrar respuestas en la comunidad web (fuente principal de resolución de dudas junto con los profesores). Siempre se ha tenido el apoyo de tutora y departamento para solventar los problemas, pero también ellos investigaban sobre el tema a la par que yo.

A partir de aquí, este trabajo es una puerta que abre un mundo de posibilidades a futuros trabajos en esta línea. Puede ser tomado como referencia para empezar a trabajar en distinto terrenos relacionados con los datos espaciales en entornos Big Data. Entre los más destacados se encuentran los siguientes:

- Implementación de un traductor de consultas espaciales SQL Estándar a HQL: Trabajar con PostGres a la par que con Hive ha evidenciado que es viable implementar un traductor de consultas espaciales de uno a otro medio. Parte de las diferencias entre lenguajes se han extraído en este trabajo, pero sería necesario realizar un estudio sintáctico más profundo.

- Implementación de un asistente completo de consultas espaciales para Hive: Es una de las futuras líneas más evidentes establecidas con este trabajo. Este nuevo proyecto estaría más enfocado en personas sin ningún conocimiento en SQL ni en datos

espaciales, y que necesiten de forma rápida implementar búsquedas sin conocer la tecnología que hay debajo.

- Estudiar más en profundidad combinaciones espaciales con cláusula ON: Uno de los puntos críticos en este trabajo ha resultado ser las combinaciones mediante funciones y datos espaciales. En este trabajo se han propuesto soluciones para resolver JOIN espaciales, pero casi todas centradas en el filtrado de resultados con cláusula WHERE. Sería interesante seguir investigando el filtrado con ON.

- Uso de HBase como medio de importación de los datos: Como ya se adelantaba en las alternativas de diseño, HBase es un medio perfecto para realizar importación de datos. El hecho de que esté integrado con Hadoop al igual que Hive hace que pueda ser apoyo para la carga de datos. Esto se debe a que mediante Hive y HBase puede accederse a las mismas tablas, ya que ambas herramientas trabajan sobre HDFS. Un ejemplo de esta implementación es la proporcionada por un destacado desarrollador de Microsoft en su blog *(7. Bibliografía, Otras fuentes de interés).*

- Implementar en Java bibliotecas nuevas que no existen en Hive y sí en PostGIS: Como se ha visto a lo largo de este trabajo existen funciones espaciales sin una implementación para Hive. Podría implementarse un equivalente de las funciones de PostGIS para este sistema.

- Estudiar Hive en profundidad: arquitectura y funcionamiento (*Map Reduce*): En la memoria no se ha profundizado en detalle en el funcionamiento y estructura de Hive ni HDFS. Un buen complemento a este trabajo sería un estudio sobre la herramienta, haciendo hincapié en cómo se ejecutan por debajo las consultas en Hive y cómo funciona el *Map Reduce*.

- Implementar más consultas: En este proyecto se han desarrollado consultas en base a un caso práctico concreto, actuando sobre dos tipos de datos espaciales. Aunque el esquema de creación de consultas no varíe, con el uso de otras funciones espaciales sobre otros tipos de datos espaciales aumentaría la riqueza del estudio.

- Investigar acerca de la eficiencia en la ejecución de consultas: Este trabajo se ha centrado en conseguir elaborar consultas que funcionen. Aunque se haya buscado la manera de construir búsquedas eficientes, no ha sido el principal objetivo.

Son interesantes y abundantes las vías que Hive abre en el mundo de las telecomunicaciones. Su facilidad de uso gracias a HQL y su velocidad le sitúan como uno de los grandes referentes del NoSQL.

5. Aspectos legales

Este punto recoge los aspectos de tipo legal que se han tenido en cuenta para usar las distintas herramientas de trabajo y las API de programación, así como en el uso del *data set* de la tabla *Uber*. Se hará énfasis en por qué se han utilizado dichas herramientas y dichas APIs y no otras en función de sus términos legales. En el caso de los datos de la tabla *Uber* se describe el marco legal que se ha respetado para su utilización en este proyecto.

5.1. Herramientas de trabajo

El desarrollo del caso práctico se ha realizado usando *Hortonworks Data Platform* (*Anexo II* y *Anexo III*). Esta plataforma contiene una serie de productos integrados, probados y combinados, donde las versiones de cada uno de estos productos han sido elegidas teniendo en cuenta que deben de ser lo más estables y compatibles entre sí (Java, Python, etc.). La plataforma y los productos que la forman están provistos de licencia de código abierto, por lo que la razón fundamental por la que se ha utilizado este software es poder descargarlo y usarlo de forma gratuita , existiendo sobre éste un soporte (documentación y tutoriales) que permite el acceso al uso de Big Data de forma sencilla. Para conocer todos los productos que contiene la plataforma, tanto propios como de *third-party*, puede consultarse la lista que proporciona Hortonworks donde se indican las versiones de todos ellos y bajo qué licencia están [11].

Citando a Hortonworks en su web, se constata que Hadoop es una plataforma de libre uso, y de todos los componentes que la forman:

"Hortonworks is dedicated to providing a 100% open source, Apache licensed product for our users and community. A variety of open source projects have been integrated, tested and combined as part of the Hortonworks Data Platform (HDP). The specific versions of these projects have been selected because they have been tested and certified to be the most stable, compatible and production-ready releases available in their respective domains."[11]

La mayoría de productos de Hortonworks están bajo licencia de Apache 2. A su vez, los productos Apache están bajo licencia libre, y permiten el uso, reproducción y distribución de su software según la normativa que presenta en su web. [12].

QGIS (*Anexo III*) es también una herramienta que está bajo licencia libre, en concreto licencia *Creative Commons*. Los términos de licencia libre en este caso giran en torno a la modificación y a la compartición del software. Se puede copiar o distribuir el software de forma totalmente libre, y por otro lado se puede transformar y modificar a gusto de usuario [13].

PostGres tiene su propia licencia gratuita, lo cual quiere decir que no hace uso de otras licencias más comunes como GNU o *Creative Commons.* La licencia permite el uso, copia, modificación y distribución del software y su documentación para cualquier fin y sin firmar acuerdos de garantía [14].

Como puede verse, el rasgo común que caracteriza a todas las licencias aquí citadas es que son de uso libre. A parte de los aspectos técnicos de cada una y de lo que se pueda hacer con cada

herramienta, el coste de uso cero ha sido uno de los factores que han llevado a que se usen para este trabajo dichas herramientas.

5.2. APIs y lenguajes

Existen aspectos legales y técnicos que han regido el uso que se ha dado a los lenguajes de programación utilizados, así como a las APIs espaciales tanto de PostGres como de GItHub para Hive.

Para este proyecto se debía usar HQL como lenguaje de consultas sobre Hive. Este lenguaje está reglamentado y documentado en el manual de consulta de Hive que proporciona Apache. En la web de Apache se puede consultar también el tipo de licencia que presenta el lenguaje, el cual en este caso es una licencia de software libre, y es proporcionada por Apache. Esta licencia se la conoce como la *Atlassian Confluence Open Source Project License*.

PostGIS ha sido la API de referencia a partir de la cual se han diseñado y construido las consultas espaciales en Hive, puesto que su homólogo en los repositorios de GitHub son en esencia idénticos en funcionalidad y uso. Esta API propia de PostGres está bajo licencia GNU.

Las APIs espaciales de *GitHub*, escritas en lenguaje *Java*, son la base técnica de este trabajo. El repositorio de *GitHub* contiene numerosas aportaciones en forma de bibliotecas espaciales y bibliotecas de otra índole, para poder utilizarse en *Hive*. Los usuarios de *GitHub* suben sus trabajos de forma desinteresada para que cualquier desarrollador pueda descargárselos y trabajar a partir de ello. Las licencias de los trabajos pueden ser editadas por los desarrolladores, por lo que puede haber APIs con licencias dispares, abiertas o cerradas, o incluso no disponer de una. En el caso de las APIs utilizadas en este trabajo (*esri-geometry-api* y *spatial-sdk-hive*) son de código abierto.

Al igual que con las herramientas de trabajo, el uso de las APIs espaciales cuya fuente es *GitHub*, y el uso de PostGIS y HQL está justificado no sólo por ser objetivo de estudio del trabajo de este proyecto, sino por su orientación *open source* y su fácil accesibilidad.

5.3. *Data set* de *Uber* (fichero *Uber.csv*)

El caso práctico que se ha diseñado para la realización de este proyecto gira en torno a la utilización del fichero *Uber.csv*. Dicho fichero contiene puntos geográficos registrados a partir de cuentas de usuario de la aplicación Uber. La información incluida en el fichero en ningún caso recoge información personal de los usuarios que registraron sus rutas, puesto que dicha información fue suprimida por Hortonworks para evitar problemas legales. Esto en cuanto a España es una práctica correcta para no violar la Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal. En concreto, dos de los artículos de esta ley española son de especial relevancia en este caso, el cinco y el seis concretamente [15]. El artículo cinco se titula *Derecho de información en la recogida de datos*, y en su primer punto dice así:

1. Los interesados a los que se soliciten datos personales deberán ser previamente informados de modo expreso, preciso e inequívoco:

a) De la existencia de un fichero o tratamiento de datos de carácter personal, de la finalidad de la recogida de éstos y de los destinatarios de la información.

b) Del carácter obligatorio o facultativo de su respuesta a las preguntas que les sean planteadas.

c) De las consecuencias de la obtención de los datos o de la negativa a suministrarlos.

d) De la posibilidad de ejercitar los derechos de acceso, rectificación, cancelación y oposición.

e) De la identidad y dirección del responsable del tratamiento o, en su caso, de su representante.

Que los ficheros no incluyan información personal de ninguno de los usuarios de Uber de cuyas rutas se utilizaron en *Uber.csv* hace que no se tenga que avisar a todas las personas con datos en este fichero de la utilización de su información personal para este trabajo académico.

El artículo seis por otro lado se titula *Consentimiento del afectado*, y sigue la misma línea que el cinco. En este caso también el primer punto es de interés:

1. El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa.

6. Gestión del proyecto

Este punto refleja el trabajo de seguimiento y estimación de recursos que se ha considerado para este proyecto, en concreto la planificación en el tiempo y la estimación de presupuesto.

En la planificación se muestra la distribución en el tiempo de las distintas tareas de las que ha constado el proyecto, además de explicarse la función de dichas tareas. Para ver cómo de buena ha sido la estimación inicial se incluyen dos diagramas de *Gantt*: uno que muestra la estimación previa al comienzo del trabajo, y otro que refleja el coste real en jornadas del proyecto. Al final se incluyen comentarios sobre el transcurso del trabajo.

En el cálculo presupuestario se explica el desglose de costes que surgirían en el proyecto de ser éste desarrollado en el ámbito empresarial. Se incluyen tanto los costes de personal como los costes materiales que genera la actividad en el trabajo.

6.1. Planificación del trabajo

El proyecto documentado en esta memoria ha sido realizado en base a una planificación inicial ideal realizada antes de comenzarlo. Dicha planificación ha consistido en descomponer el trabajo a realizar en tareas y sub tareas, de tal forma que se consiguiera obtener períodos medibles. Para cada tarea y sub tarea se estimó de inicio un tiempo de realización, se les dio funciones a esas tareas, y a partir de esa información se construyó un diagrama de Gantt.

Las tareas principales en las que se ha dividido el *planning* del proyecto han sido las siguientes:

- Estudio previo/Estado del arte: Primera fase del trabajo. En ella se estudian proyectos de índole similar que se hayan presentado anteriormente en la Universidad para tomarlos como referencias. Se analiza y comprende el contexto tecnológico en el que el caso práctico se enmarca, así como los fundamentos teóricos acerca de las tecnologías utilizadas, prestando atención a los datos espaciales y el NoSQL dentro del movimiento Big Data. Se evalúan además distintas alternativas de solución.

- Caso práctico: Segunda fase del proyecto. En ella se diseña un caso práctico para resolver el problema planteado y se lleva a cabo.

- Experimentación: Tercera Fase del proyecto. Consiste tanto en la experimentación en *Hive*, así como en la elaboración del esquema de consultas y la explicación de sus ramas y ejemplos prácticos.

- Pruebas/conclusiones: Última fase del proyecto. Consiste en las pruebas de las consultas espaciales más relevantes, así como en el establecimiento de resultados finales (esquema de decisiones), conclusiones, alternativas de solución y futuros trabajos.

La planificación del proyecto consiste en una sucesión en cascada de tareas, donde para realizar la inmediatamente posterior debe de haberse completado la anterior. En el caso de la tarea relacionada con el caso práctico no se sigue un proceso secuencial, puesto que la

experimentación con las consultas y la variación constante de los parámetros del caso práctico dan lugar a múltiples cambios que hacen que se deba rehacer el trabajo realizado. Para simplificar el *planning* se ha decidido incluir la planificación de la tarea que engloba el caso práctico de forma lineal, en cascada.

Las tareas y sub tareas del proyecto así como los tiempos planificados para cada una de ellas están reflejados en la Tabla 45.

Tabla 45: Tareas, sub tareas y estimación

Tareas	Sub tareas	Descripción	Estimación inicial (Horas)	Tiempo real (Horas)
Estudio previo/Estado del arte	Estudio de trabajos previos/similares	Búsqueda y lectura de proyectos que hayan trabajado en el mismo campo sobre el que se desarrolla el trabajo, que puedan servir como base informativa o técnica.	33	33
	Estudio de *Big Data*, *NoSQL*, bases de datos espaciales, *Hive*	Elaboración del estado del arte. Búsqueda de información en fuentes veraces. Elaboración de contexto a partir de dicha información.	99	99
	Estudio de entornos de trabajo	Configuración y creación de entornos de trabajo, y primera toma de contacto con *Hive* y *Hadoop*.	39	27
	Análisis de la solución	Estudio de la solución a llevar a cabo una vez realizada la experimentación.	15	30
Caso práctico	Diseño y creación del caso práctico	Elaboración de un caso práctico acotado que sirva de marco de trabajo: tablas, datos, importaciones, etc.	69	69
Experimentación	Diseño y ejecución de batería de consultas	Primera toma de contacto con consultas y datos espaciales en *Hive*. Elaboración de distintos casos de búsqueda de información en las tablas propuestas para el caso práctico, y a partir de ellos elaboración de una batería completa de consultas.	75	75
	Establecimiento de esquema de consultas	A partir de las consultas obtenidas, estudio de posibles vías de resolución de consultas espaciales y establecimiento de patrones comunes de diseño.	57	57
Pruebas/Conclusiones	Prueba de consultas espaciales	Prueba de las consultas más relevantes de entre las contenidas en la batería. Uso de pruebas de caja negra/blanca según la consulta.	36	36

	Establecimiento de resultados finales y conclusiones	Resultados finales. Conclusiones acerca del trabajo realizado y crítica sobre los resultados obtenidos. Identificación de distintas formas en las que ciertas partes del trabajo podrían haberse hecho de distinta manera, y apertura de nuevas vías de trabajo a partir de los resultados del caso práctico.	54	72

El proyecto se estimó que se realizaría a lo largo de 7 meses (Noviembre 2014 – Junio 2015). Las horas estimadas y reales se han calculado a partir de las jornadas establecidas en los diagramas de *Gantt*. Se ha supuesto que los días festivos y fines de semana también se han trabajado, y que se han dedicado tres horas diarias a la realización del proyecto. Por lo tanto, el cálculo de las horas para cada sub tarea queda establecido como el producto entre las horas dedicadas diarias y las jornadas que duran cada sub tarea.

La planificación inicial (Ilustración 30) estimaba una duración general del proyecto de 160 días. Las tareas más pesadas son las relacionadas con el estado del arte y la experimentación de consultas espaciales. La tarea más crítica es el análisis de la solución, como se verá en el diagrama de tiempo real. La tarea de pruebas y conclusiones no es fácil de predecir su duración.

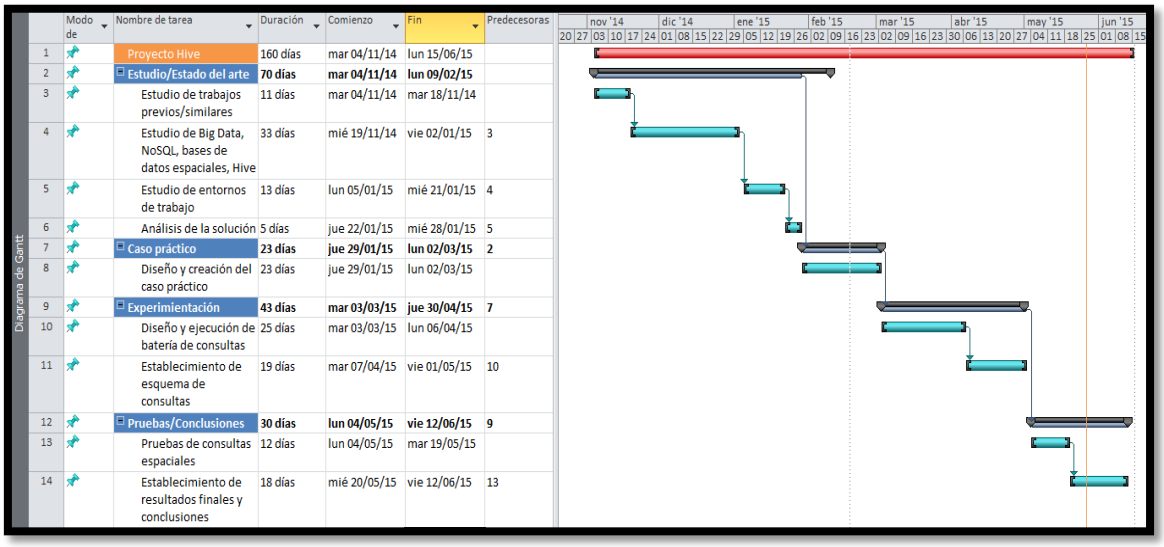

Ilustración 30: *Gantt* de planificación inicial

Uno de los puntos críticos del proyecto ha sido elegir qué aplicación práctica se daba a la experimentación realizada (Tarea Análisis de la solución). Al haber habido varios giros sobre el tema, esta tarea se alargó en el tiempo produciendo un retraso en el global del proyecto. Por otro lado, el establecimiento de los métodos de prueba y la documentación de las mismas finalmente fue más laborioso de lo que en un principio se creía, por lo que también existe un desfase de horas en esta tarea. Se puede consultar la comparación de tiempos en jornadas en la Ilustración 31.

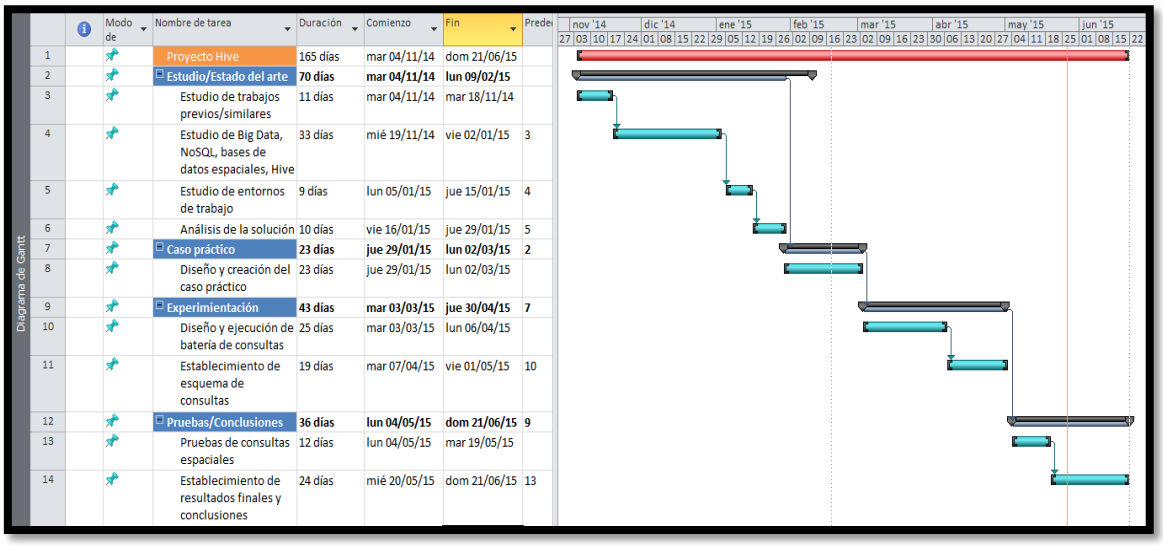

Ilustración 31: *Gantt* de planificación real

6.2. Cálculo y estimación de presupuesto

El proyecto que se ha explicado y detallado a lo largo de esta memoria no tiene ningún fin lucrativo. No obstante, es necesario incluir un estudio presupuestario para entender el coste del proyecto si este se desarrollara en un entorno comercial. El trabajo documentado en esta memoria es un modo práctico e iniciativo para comenzar a trabajar en uno de los paradigmas más incipientes en la actualidad, los *Big Data*, por lo que es probable que alguna empresa pueda interesarse por él y desee llevarlo a cabo.

Se explicarán a continuación los costes tanto personales como los materiales implicados en la realización del trabajo, así como otras valías adicionales. Se tendrá en cuenta para dichos cálculos que el proyecto es llevado a cabo por una empresa propia. También se incluirá la imputación de impuestos. Las cifras numéricas se proporcionarán en Euros, moneda oficial de la Unión Europea.

En primer lugar, el proyecto conlleva unos gastos de personal. Se ha considerado que el gasto por hora del personal es de 16,5€. No se incluye al tutor del proyecto en los recursos humanos, por lo que sólo se incluye a la persona que desarrolla el trabajo. Dicha persona desarrolla las tareas identificadas en el punto ***¡Error! No se encuentra el origen de la referencia.***. *Planificación del trabajo* en un determinado número de horas (horas estimadas), las cuales conllevan un coste asociado. La tabla 46 refleja el gasto por tarea que supondría la persona que realiza el trabajo.

Tabla 46: Gastos de personal por tarea

Tarea	Horas de trabajo (h)	Coste (€)
Estudio previo/Estado del arte	186	3069
Caso práctico	69	1138.5
Experimentación	132	2178
Pruebas/Conclusiones	90	1485
TOTAL	**477**	**7870.5**

Otro de los gastos a contemplar son los relacionados con los medios materiales utilizados para la realización del proyecto (Tabla 47). Los medios que se han tenido en cuenta en el presupuesto son medios informáticos, pudiendo ser estos de tipo software o hardware. Para cada medio se ha indicado su coste total en el mercado, así como el coste real de uso del mismo, calculado a partir de la amortización que se ha estimado, y proporcional a la duración en meses del trabajo. En definitiva, el coste de cada medio queda reducido al tiempo de uso que se le ha dado. En el caso de ser medios sin coste alguno no se ha tenido en cuenta ninguno de estos conceptos.

Tabla 47: Gastos materiales

Tipo	Recurso	Coste (€)	Amortización (meses)	Amortización (€/mes)	Período de uso (meses)	Coste (€)
HW	Toshiba Satellite l50-b-23g	589.00	60	9.82	7	68.74
	PC Sobremesa PCComponentes	669.00	60	11.15	7	78.05
SW	Microsoft Office 2010 Home and Student	269.00	120	2.24	7	15.68
	Sistema Operativo Windows 8.1 (licencia estudiante)	0	-	-	-	0
	Sistema Operativo Windows Server 2008 (licencia estudiante)	0	-	-	-	0
	Hadoop 1.3 (*open source*)	0	-	-	-	0
	PostGreSQL 9.3 (*open source*)	0	-	-	-	0
Total						**162.47**

Los gastos directos, calculados a partir de la suma entre gastos de personal y materiales, ascienden a la cifra de 8032.97 €. Sobre estos gastos, se ha supuesto unos gastos indirectos como la luz, el agua o el gas del 4%. En la tabla 52 se indica el coste total real, teniendo en cuenta gastos indirectos e impuestos (IVA) (Tabla 48).

Tabla 48: Gastos totales

Gastos	Coste (€)
Directos	8032.97
Indirectos	401.65
Total antes de impuestos	8434.62
IVA 21%	1771.27
Total (IVA 21%)	**10205.89**

7. Bibliografía

Referencias

[1] R. Barranco. "¿Qué es Big Data?", *ibm.com*, 18-06-2012. [En línea]. Disponible en: https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/ [Consulta: Diciembre 2014].

[2] R.Herranz. *Bases de datos No-SQL. Arquitectura y ejemplos de aplicación.*, 2014.

[3] Tom Plunkett, *Oracle Big Data Handbook.* Oracle Press, 2014. [E-book] Disponible en: http://proquest.safaribooksonline.com/book/databases/business-intelligence/9780071827263.

[4] Microsoft. "Propiedades ACID", *msdn.microsoft.com*. [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/aa719484%28v=vs.71%29.aspx [Consulta: Diciembre 2014].

[5] Cisco. "Internet será cuatro veces más grande en 2016", *cisco.com*, 30-05-2012. [En línea]. Disponible en: http://www.cisco.com/web/ES/about/press/2012/2012-05-30-internet-sera-cuatro-veces-mas-grande-en-2016--informe-vini-de-cisco.html [Consulta: Diciembre 2014].

[6] MongoDB. "NoSQL Database Explained", *mongodb.com*, 2015. [En línea]. Disponible en: http://www.mongodb.com/nosql-explained [Consulta: Diciembre 2014].

[7] R. Harmut, "An Introduction to Spatial Database Systems", *VLDB Journal, Vol. 3, nº 4, pp 1-3*, Septiembre 1994 [En línea]. Disponible en: http://www.dpi.inpe.br/geopro/referencias/guting_spatialdbms.pdf [Consulta: Diciembre 2014]

[8] Manuel Martín Martín, *Manual PostGIS.* [E-book] Disponible en: http://postgis.refractions.net/documentation/postgis-spanish.pdf

[9] Apache. "Hive Data Types", *cwiki.apache.org*, 04-05-2012. [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/Home [Consulta: Enero 2015].

[10] Apache. "Hive Data Definition Language", *cwiki.apache.org*, 13-06-2015. [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL [Consulta: Enero 2015].

[11] Hortonworks. "Hortonworks Data Platform, Optional Add-Ons and 3rd-Party Component Licenses", *hortonworks*. [En línea]. Disponible en: http://hortonworks.com/products/licenses/ [Consulta: Marzo 2015].

[12] Apache License, The Apache Software Foundation V2.0-2004. [En línea]. Disponible en: http://www.apache.org/licenses/LICENSE-2.0

[13] Attribution-ShareAlike 3.0 Unported, Creative Commons V3.0. [En línea]. Disponible en: http://creativecommons.org/licenses/by-sa/3.0/

[14] PostgresSQL License, The PostgreSQL Global Development Group. [En línea]. Disponible en: http://www.postgresql.org/about/licence/

[15] ESPAÑA, 1999. Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal. *Boletín Oficial del Estado* [en línea]. 14 dic 1999, Artículos nº 5 y 6. Disponible en: http://www.boe.es/buscar/act.php?id=BOE-A-1999-23750&tn=1&p=20110305&vd=#a1 [Consulta: Marzo 2015].

Otros enlaces de interés

Apache Hadoop: http://wiki.apache.org/hadoop

Fichero *Uber.csv*: http://es.slideshare.net/hortonworks/hive-meetup-spatial-1

Fichero *districts.xlsx* http://blogs.mathworks.com/loren/2014/09/06/analyzing-uber-ride-sharing-gps-data/

Ejemplo de importación de datos de Hortonworks: http://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-hive/

Importar datos en Hive desde HBase: http://azure.microsoft.com/es-es/documentation/articles/hdinsight-hbase-get-started/

Biblioteca jar con GeomFromText: http://www.java2s.com/Code/Jar/s/Downloadspatialsdkhive101sourcesjar.htm

Anexo I: Instalación y configuración de PostGres (Objetos espaciales)

PostGres es un sistema gestor de base de datos relacional cuyo código es distribuible de forma libre. Nació como un proyecto universitario, obra de Michael StoneBreaker de la universidad de Berkeley en 1986, hasta convertirse en uno de los sistemas más utilizados en la actualidad como base de almacenamiento de datos. Para dar soporte a múltiples conexiones, PostGres tiene arquitectura cliente-servidor, proporcionando mediante el uso de hilos de ejecución la posibilidad de mantener varias sesiones de usuarios abiertas a la vez, imitando el funcionamiento de otros sistemas gestores como Oracle.

Para la instalación de PostgreSQL es necesario cumplir con algunos requisitos en cuanto al sistema operativo. Se puede descargar el instalador de la aplicación desde esta web:

http://www.enterprisedb.com/products-services-training/products/postgresql-overview&ls=SEM?gclid=CjwKEAjw1ZWgBRD-n6ew0oan1xwSJABAbf8pGwfhhN_NjZ8V8sp9EIg5TIGgfkWvHkVXTnowljwLGxoCo9jw_wcB

El paquete de instalación puede descargarse para los siguientes sistemas:

- Windows de 32 y de 64 bits.
- Linux de 32 y de 64 bits.
- Mac OS X.

Para la elaboración de este caso práctico se ha descargado e instalado la versión para Windows de 64 bits. La instalación para Windows de 32 bits es idéntica a lo explicado en este anexo. Para más información sobre cómo instalar el gestor de bases de datos en otros sistemas que no sean Windows consultar el manual de instalación completo indicado en el siguiente enlace:

http://www.enterprisedb.com/docs/en/9.3/pginstguide/Table%20of%20Contents.htm

La versión de PostGres que se ha utilizado es la 9.3.4-3. Una vez descargado el instalador se procede a abrirlo. La instalación de este entorno de trabajo no conlleva una dificultad elevada, gracias al intuitivo instalador que proporciona la herramienta. El asistente preguntará en qué directorio se desea instalar el programa. Una vez escogido dicho directorio, el programa se instalará correctamente.

Entre los programas que se han instalado, son importantes *PgAdmin* y *Applicattion Stack Builder*. El primero, *PgAdmin*, es la interfaz gráfica con la que se puede interactuar con PostGres de manera intuitiva, ya sea a la hora de crear nuevos espacios de trabajo o editar nuestros propios scripts de desarrollo. La segunda herramienta, *Application Stack Builder*, permite de forma muy sencilla añadir paquetes adicionales a PostGres si es que es necesario incorporarlos. Para este caso práctico en el que se necesita trabajar con datos espaciales, es necesario que antes de empezar a trabajar con PostGres se añada el paquete de PostGIS, el cual proporciona bibliotecas para trabajar con datos espaciales. Al iniciar el *Application Stack*

Builder, en el asistente de instalación de extensiones se debe seleccionar los paquetes de *Spatial Extensions*, tal como aparece en la Ilustración 32.

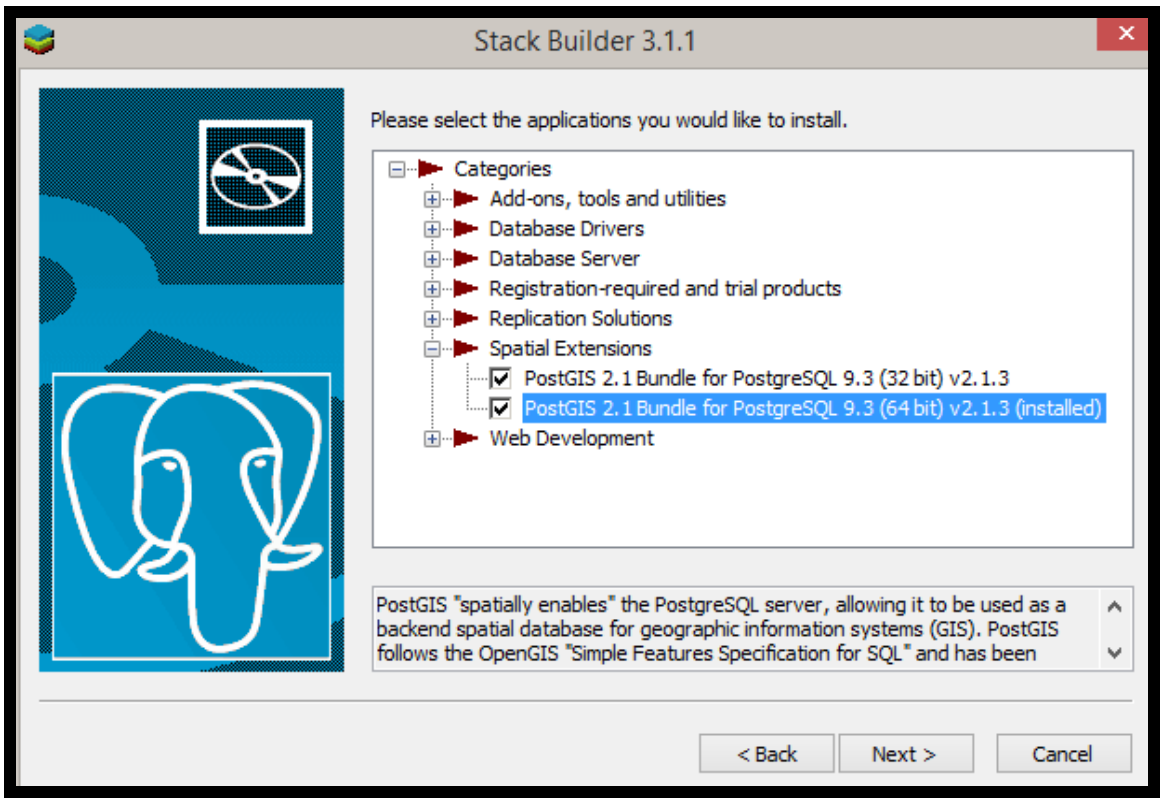

Ilustración 32: Adición de paquetes espaciales PostGIS para PostGres

Con esta configuración, ya se puede iniciar el *PgAdmin*. Es necesaria una última configuración antes de poder trabajar con objetos espaciales en PostGres, la cual se explicará a continuación.

Se explica brevemente ahora cómo crear un nuevo esquema de datos. Desde el *PgAdmin* se crea una conexión de bases de datos y una base de datos valga la redundancia. Se puede considerar como servidor el que viene por defecto, PostGreSQL 9.3, y el esquema de datos en este caso se ha nombrado como "TFG_Database". Se puede observar esta configuración en la Ilustración 33.

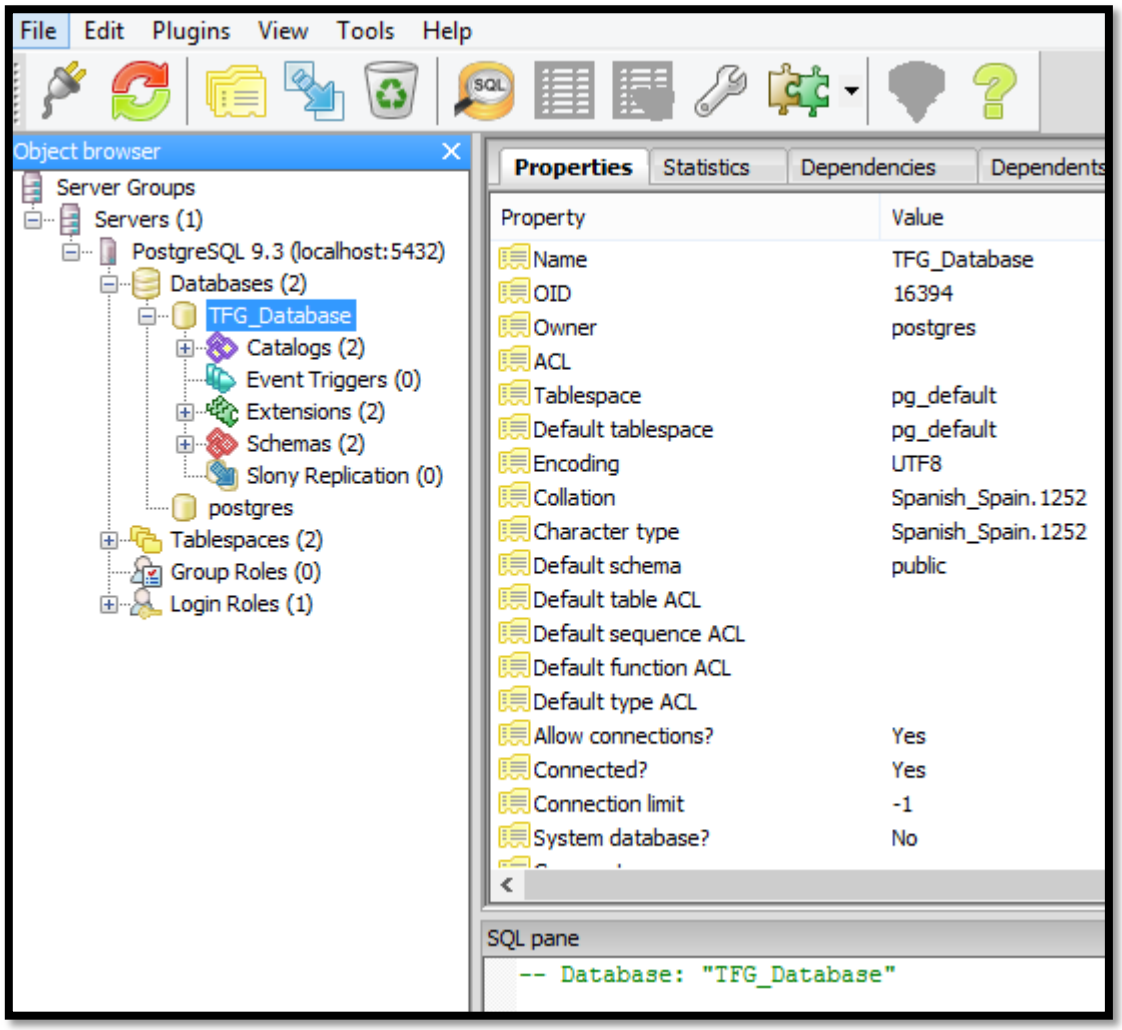

Ilustración 33: Interfaz de PostGres con la configuración inicial de conexión y base de datos

Ahora se procede a abrir un editor SQL. Para ello se pulsa en la barra de herramientas superior sobre el botón SQL, teniendo previamente seleccionado como base de datos la creada con anterioridad, tal como se ve en la Ilustración 33. Tras pulsar en el botón de SQL se abrirá el editor.

El último paso de configuración de PostGres para que se pueda trabajar con PostGIS es indicar en el editor que se habilite PostGIS en la base de datos creada. Para ello, en el editor se escribe y ejecuta el siguiente mandato:

CREATE EXTENSION postgis;

Este mandato habilita la extensión de PostGIS que se ha instalado previamente desde el *Application Stack Builder*. En el editor se debe seleccionar el texto del mandato y pulsar en la barra de herramientas sobre el icono con flecha verde, tal como se indica en la Ilustración 34 con marcado rojo.

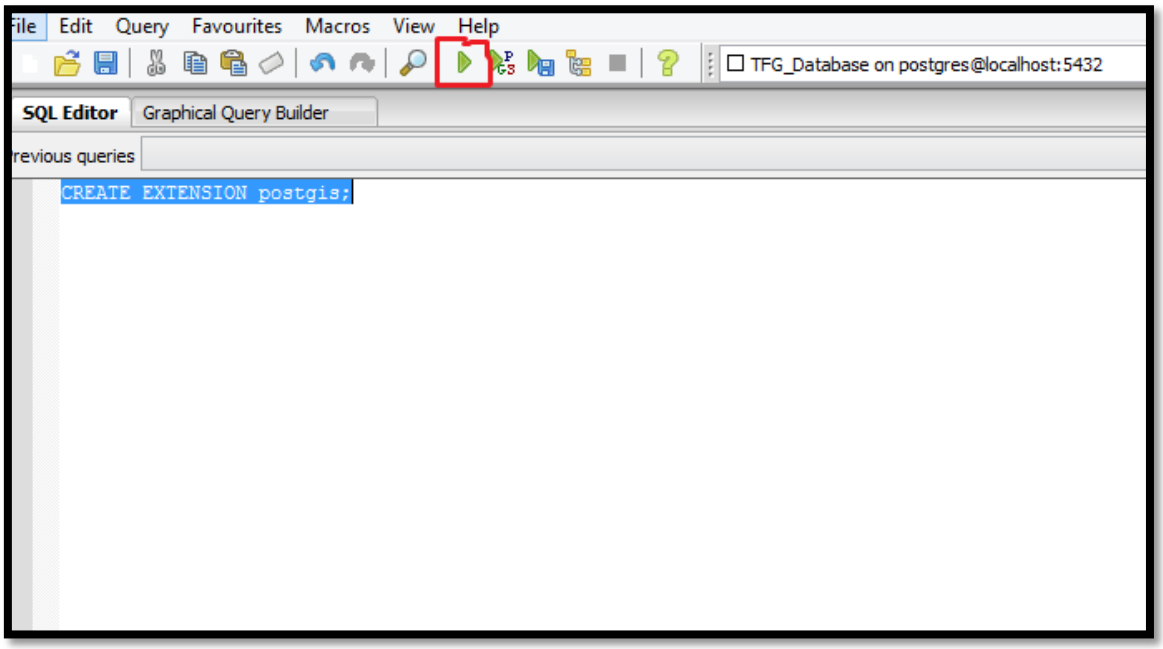

Ilustración 34: Habilitar PostGIS en la base de datos

Tras la ejecución del mandato, ya se dispondría de todo lo necesario para usar objetos espaciales en PostGres.

Anexo II: Instalación y configuración de Hive en Windows con SandBox

Esta instalación provee la forma más accesible para manejar Hive. Consiste, por un lado, en una máquina virtual Red Hat, que al instalarse configura todos los parámetros necesarios para poder trabajar con la herramienta. Por otro lado, desde un navegador web en Windows y con la dirección local que proporciona la instalación cuando esta finaliza en la máquina virtual, se accede a una intuitiva interfaz de manejo de Hive. La ventaja de este entorno es que no necesita configuraciones de red y datos, por lo que es ideal para iniciarse en Hive.

La descarga de la máquina virtual se realiza desde la web de Hortonworks.En el caso de este caso práctico, se ha utilizado la versión para VirtualBox. Para más información sobre cómo instalar o configurar parámetros de un servicio virtualizado de VirtualBox en Windows, ver el manual: https://www.virtualbox.org/manual/UserManual.html .

En el siguiente enlace: http://hortonworks.com/products/hortonworks-sandbox/#install se pincha sobre el botón correspondiente a la versión de VirtualBox y comenzará a descargarse la máquina virtual. El archivo descargado tiene como nombre "Hortonworks_Sandbox_2.1.ova". Una vez descargada, se procede a importar el servicio virtualizado. Para ello, se inicia VirtualBox, se pulsa en *Archivo→Importar servicio virtualizado*, y haciendo uso del explorador de archivos, se indica el archivo origen para generar la máquina virtual, es decir, el archivo descargado previamente. Al hacer click en *Aceptar* saldrá una ventana con los datos del servicio virtualizado. Especial atención en esta pantalla a que el sistema operativo aparezca como RedHat de 64 bits, así como seleccionar una carpeta destino alojada en una partición con suficiente espacio para desplegar el disco duro virtual. A continuación se pulsa en *Importar*, y deberá aparecer el servicio importado en el menú de selección inicial. La Ilustración 35 muestra una entrada en el menú de inicio para el sistema configurado.

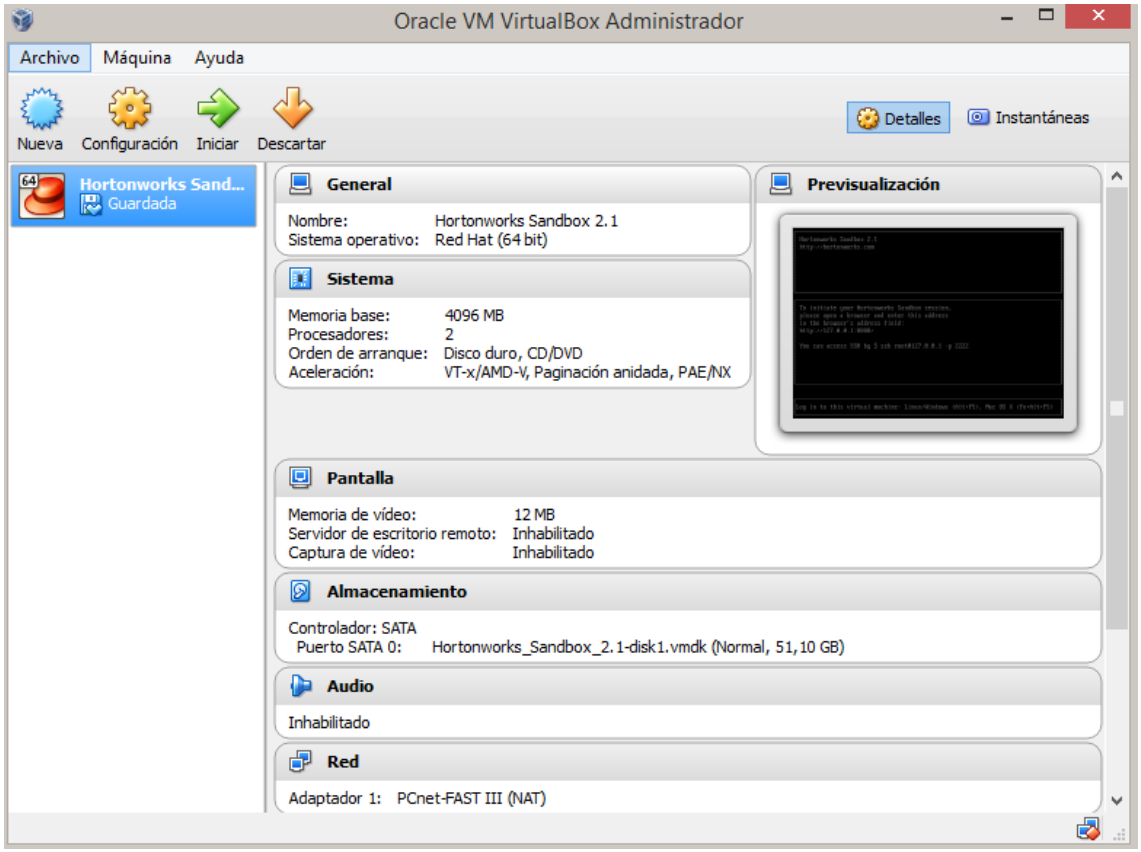

Ilustración 35: Importación correcta de máquina Red Hat en VirtualBox

Se procede a arrancar la máquina pulsando en *Iniciar*. Una vez que el sistema operativo haya iniciado y haya configurado el entorno de Hadoop y Hive entre otros, se nos muestra una URL en la consola a partir de la cual se puede acceder al Sandbox vía navegador web, tal como se ve en la Ilustración 36.

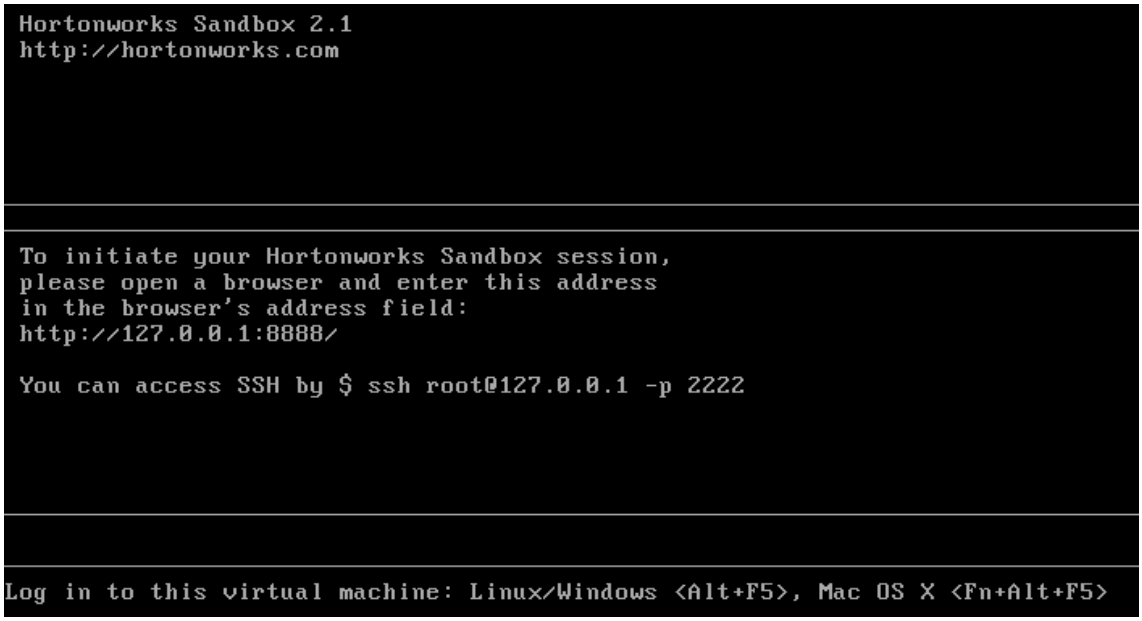

Ilustración 36: Indicación de URL para acceder a Sandbox

Sin cerrar ni apagar la máquina virtual, se ha de copiar esa URL y pegarla en la barra de direcciones del navegador. Para este caso práctico se ha utilizado Google Chrome. Se abrirá entonces un formulario de registro, donde se creará una cuenta de Hortonworks. Cuando se envíe el formulario, se accederá a la pantalla de inicio de Sandbox, donde se tendrá la opción de elegir hacer tutoriales, probar nuevas funcionalidades o acceder a Sandbox. Se accede a esta última mediante el enlace de URL local, donde se identificará entre otras opciones un icono en forma de abeja en la barra superior, perteneciente a Hive, el cual se señala en amarillo en la Ilustración 37. Si se pulsa en este icono se accederá a Hive.

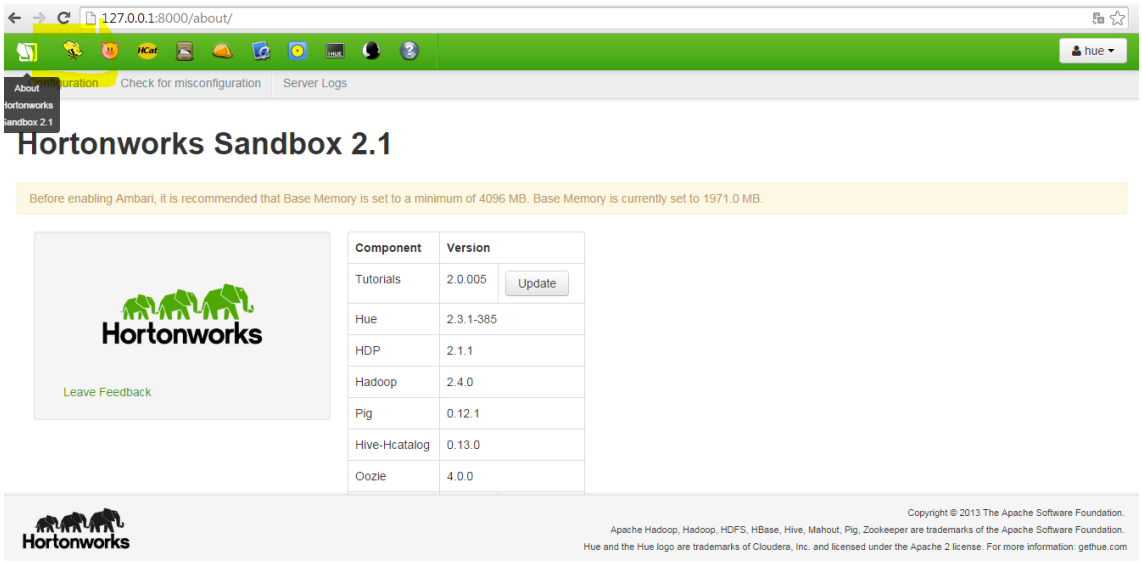

Ilustración 37: Página de Sandbox y acceso a Hive

Anexo III: Instalación y configuración de Hadoop en Windows Server

Este manual de instalación y puesta a punto explica cómo desplegar y configurar Hadoop/Hbase en un entorno real, pese a que se trabaje con repositorios locales. Decir cabe que es la instalación más compleja y con más matices de entre las cuatro que se describen en el capítulo de anexos.

Para la realización de esta instalación se ha usado de nuevo VirtualBox para crear una imagen virtual del sistema operativo sobre el que se ha trabajado, Windows Server 2008, y así evitar la instalación directa en una partición del disco duro, ganando en portabilidad. Hadoop ha sido instalado en dicho Windows Server virtualizado. En resumen, se han precisado los siguientes programas:

- **VirtualBox**
- **Windows Server 2008 R2**
- **Paquete de instalación de Hadoop**

Para saber más información acerca de VirtualBox y su instalación mirar el Anexo II. Mirar mismo anexo para saber cómo añadir un nuevo sistema operativo virtual.

Para obtener Windows Server 2008 R2 visitar la web del producto y consultar los modos de adquisición del mismo: windows.microsoft.com/es-es/windows/home .

Una vez instalada la imagen virtual del sistema operativo, se inicia la máquina. Para instalar Hadoop en Windows se precisa del paquete de instalación, el cual se puede encontrar en la página de Hortonworks http://hortonworks.com/hdp/downloads/ .En dicha página, en el apartado de "*Ready for the Enterprise*" se hace *click* en el botón de descarga de la versión para Windows Server, y con ello se descargará el paquete de Hadoop. Una vez finalizada la descarga se descomprime el archivo y dentro debe encontrarse el instalador, tal como figura en la Ilustración 38.

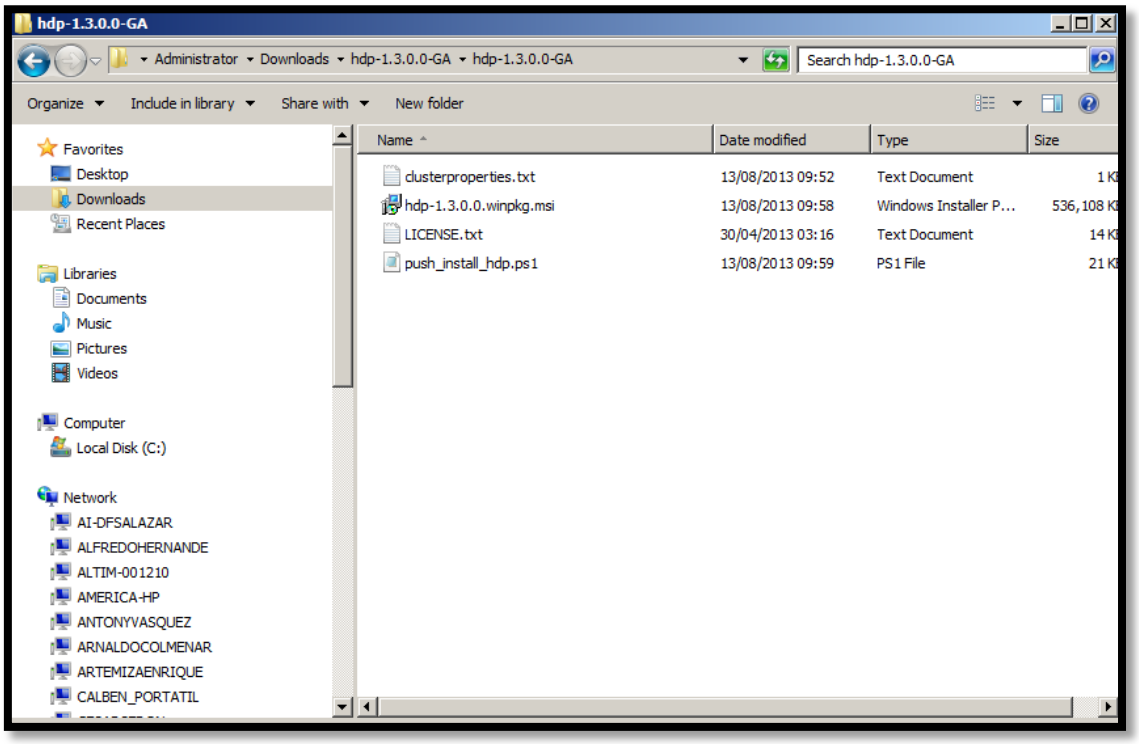

Ilustración 38: Contenido del paquete de instalación de Hadoop

A continuación se ejecuta el fichero *.msi* (*Microsoft Installer*), y comenzará la instalación. Llegados a este punto, es preciso describir cuatro configuraciones que el sistema operativo ha de cumplir de forma previa para instalar correctamente Hadoop. La versión de Windows Server que se ha usado para trabajar en este caso práctico es virgen y viene sin estas configuraciones. Tanto para esta versión de Windows Server como de otras es preciso comprobar que se tiene todo lo necesario, de lo contrario no se instalará Hadoop. Son las siguientes:

- **Instalación de Python 2.7.X (referencia) o superior más PATH (referencia)**
- **Instalación de Java JDK(referencia) más Java Home**
- **Instalación Microsoft Visual C++ (referencia) más PATH**
- **Instalación de Microsoft .NET Framework 4.0 o superior más PATH**

Python puede descargarse desde la web del fabricante. https://www.python.org/downloads/ . Una vez descargado e instalado es necesario configurar la variable de entorno PATH del sistema operativo para ubicar el directorio donde está instalado Python. Para ello se accede al Panel de control de Windows, y desde ahí se accede a *Sistema y seguridad→Sistema→Opciones avanzadas del sistema*. En la ventana emergente que saldrá se pincha en el botón *Variables de entorno*. En el listado de variables, se debe buscar PATH e indicar la ruta completa de acceso donde Python está instalado, como aparece en la Ilustración 39.

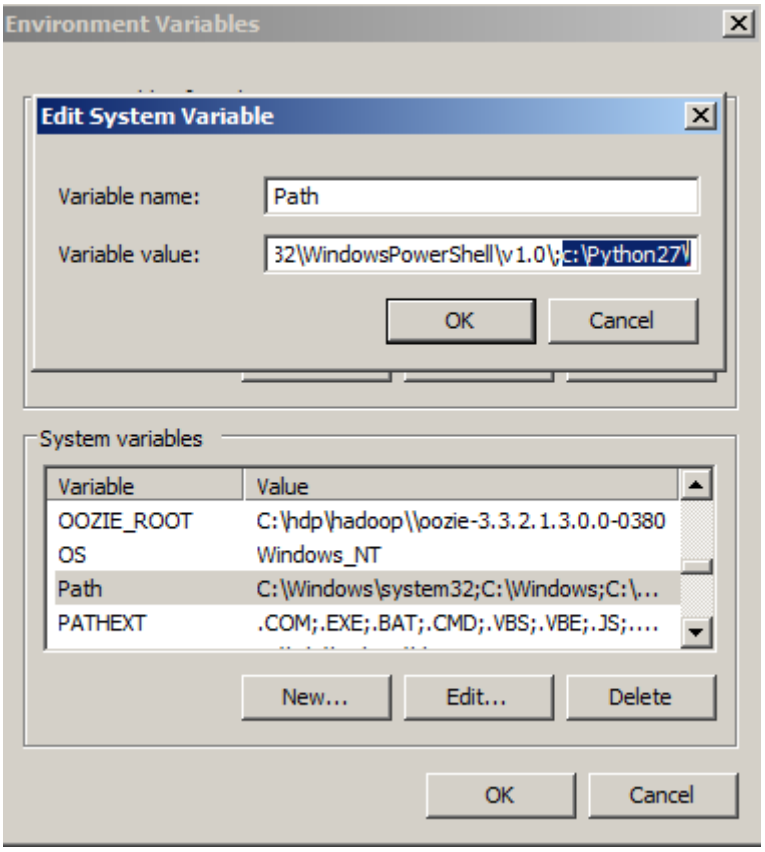

Ilustración 39: Configuración de Path para Python

Para instalar Java se descarga el instalador desde la web del fabricante https://www.java.com/es/download/. Es necesario configurar o crear, si no existe, la variable Java_Home, indicando el directorio donde está instalado Java. Puede verse esta configuración en la Ilustración 38.

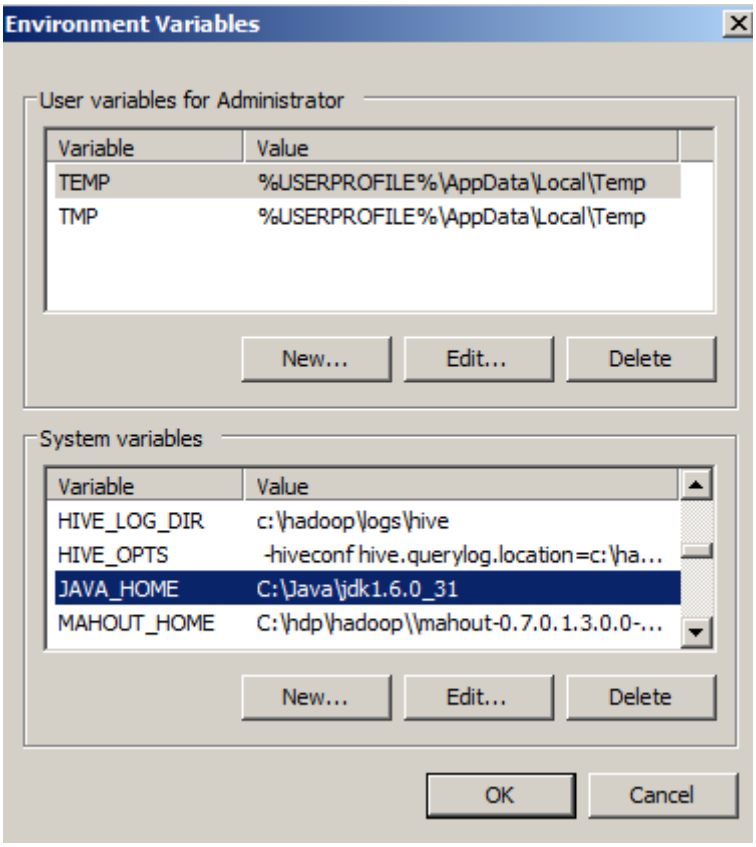

Ilustración 40: Configuración de Java_Home para Java

Para instalar Visual C++ se descarga el instalador desde la web del fabricante http://www.microsoft.com/es-es/download/details.aspx?id=30679 . No hace falta configurar en este caso las variables de entorno, puesto que durante la instalación se habrán configurado automáticamente.

En el caso de .NET Framework ya se dispone de este software en Windows Server, y no es necesaria ninguna configuración adicional.

Para más información sobre los requisitos previos a la instalación de Hadoop en Windows Server, consultar en la web de Hortonworks en esta dirección http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.7-Win/bk_installing_hdp_for_windows/content/win-software-req.html

Una vez instalado y configurado todo lo necesario previo a instalar Hadoop, se ejecuta el instalador que venía incluido en el paquete descargado desde la web de Hortonworks. Una vez arrancado el instalador, aparecerá una ventana como la de la Ilustración 41.

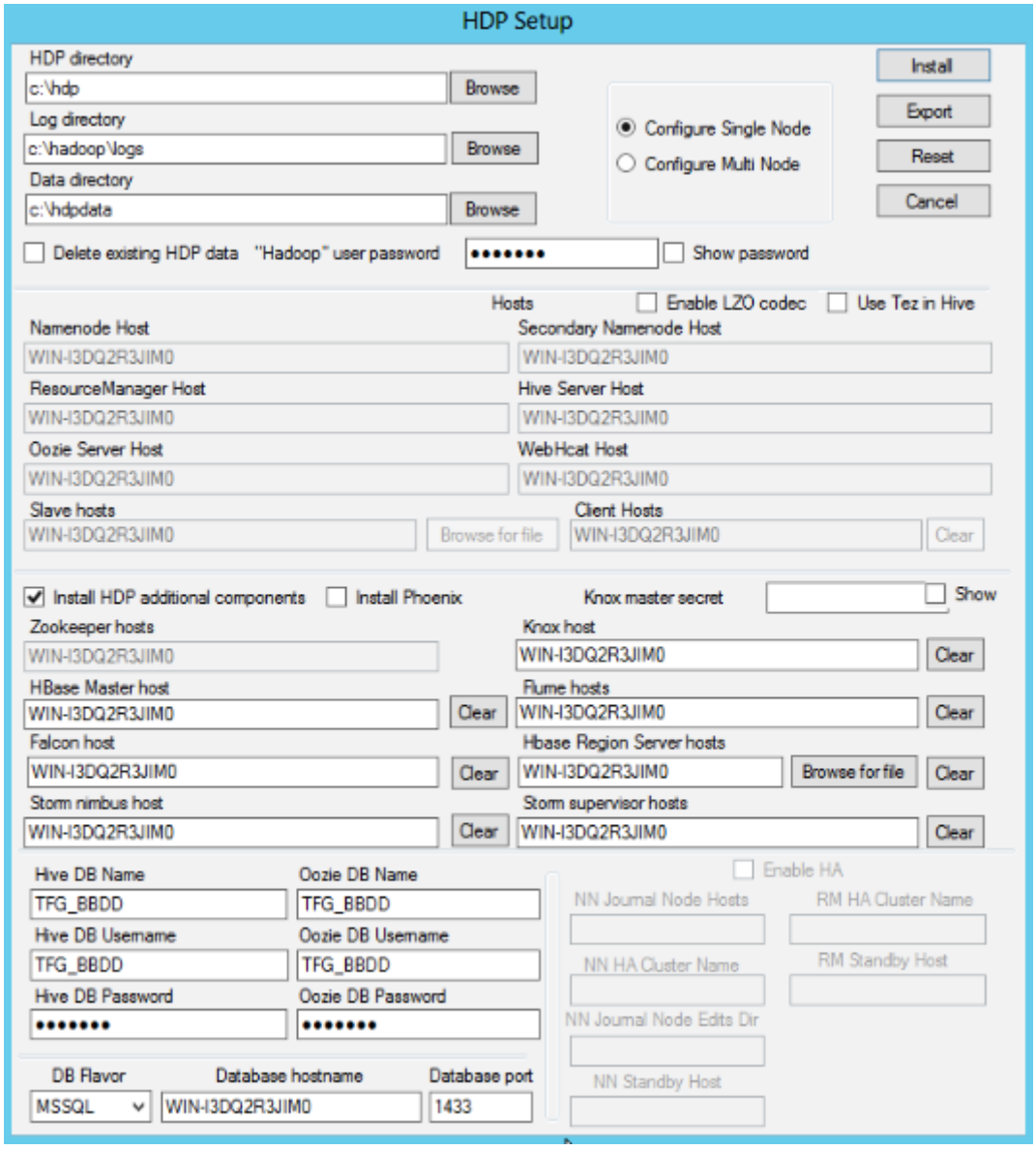

Ilustración 41: Panel de configuración de Hadoop

Se puede configurar a gusto de usuario todos los aspectos de la instalación. No obstante, es importante que se active la opción *Install HDP aditional components* para que se instale HBase. Cuando se acabe de configurar Hadoop, se pulsa en *Install* y comenzará el proceso de instalación. Si se ha configurado debidamente los requisitos del sistema la instalación debe finalizar correctamente, y en Escritorio de Windows debe aparecer el acceso directo a Hadoop Command Line, y dos archivos *.bat*, uno para Hive y otro para HBase.

Anexo IV: Instalación y configuración de QGIS

QGIS (Quantum GIS) es un sistema de Información Geográfica (SIG) de Código Abierto bajo licencia GNU. QGIS proporciona una interfaz con la cual interactuar, visualizar, gestionar, editar y analizar datos que se le proporcionen, por ejemplo como es este caso práctico, puntos geográficos.

Esta herramienta ha sido utilizada en este trabajo para visualizar de forma gráfica los resultados obtenidos de algunas de las consultas ejecutadas sobre las tablas de trabajo. En este punto se explica brevemente cómo instalar la herramienta (en plataforma Windows) y cómo hacer una carga rápida de ejemplo en QGIS.

Para instalar la herramienta es necesario descargar el paquete de instalación desde la web del fabricante a través del siguiente enlace: http://www.qgis.org/es/site/forusers/download.html . En dicha página se selecciona la versión que se desee descargar, que para el caso del trabajo práctico ha sido la versión de 64 bits de Windows para usuarios nuevos. En la Ilustración 42 puede verse las opciones de entre las que se ha de elegir para Windows.

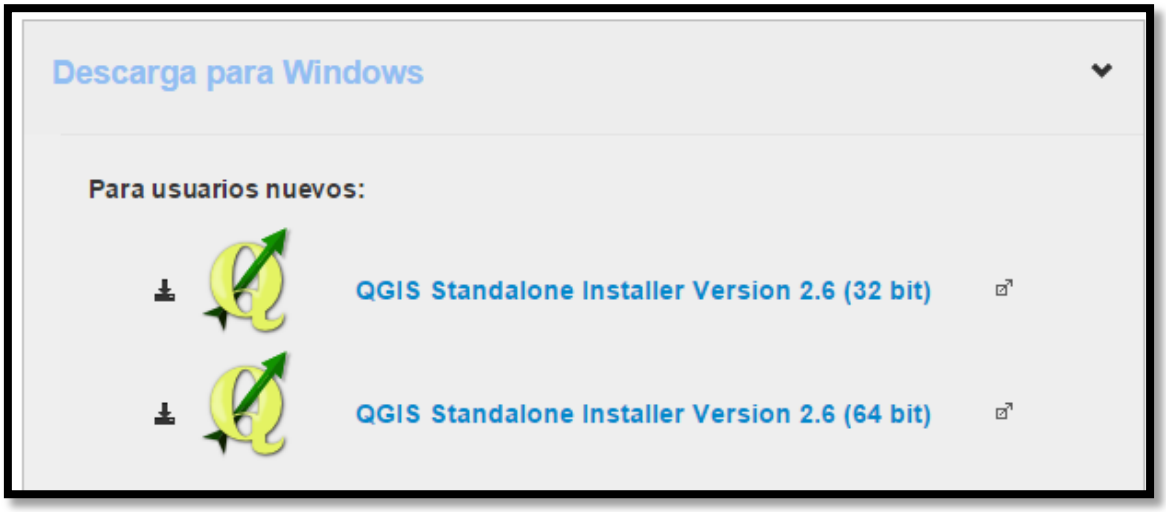

Ilustración 42: Versiones de QGIS válidas para el caso práctico

Cuando se pinche sobre una de las dos opciones se descargará el paquete de instalación, el cual debe de tener un nombre similar a "QGIS-OSGeo4W-2.6.0-1-Setup-x86_64". Una vez descargado, se procede a ejecutar dicho instalador.

El instalador nos va guiando para configurar de inicio QGIS. Se debe elegir directorio de instalación. Una vez instalado correctamente, se puede ver que se han instalado varias aplicaciones distintas. Nos interesa de entre ellas la llamada como QGIS Desktop. Si se abre, se accederá a una interfaz gráfica tal como aparece en la Ilustración 43.

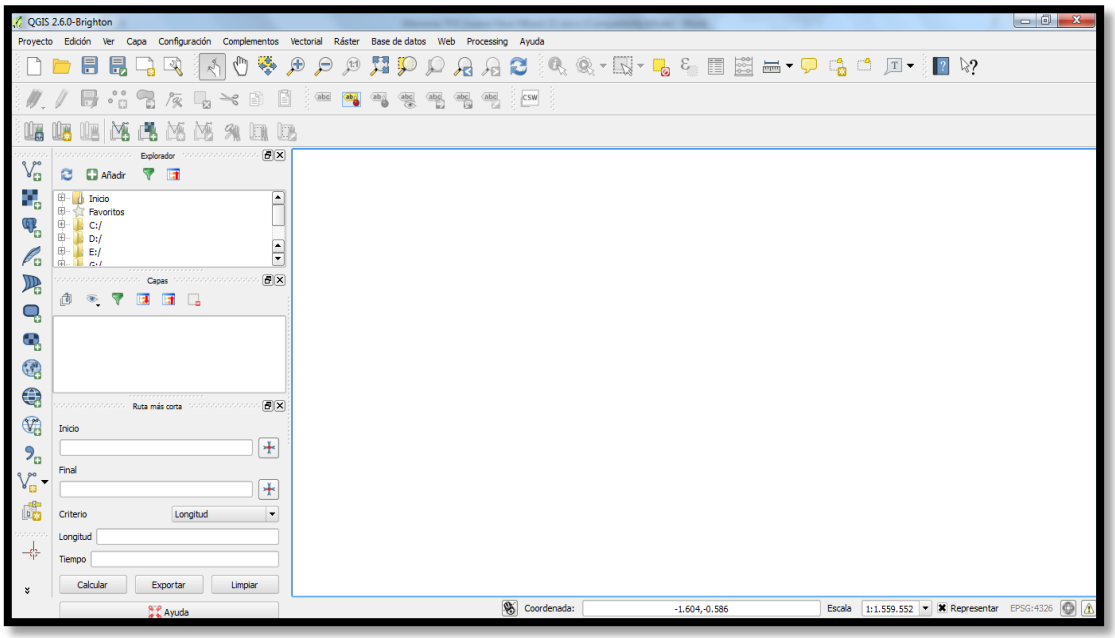

Ilustración 43: Interfaz gráfica de QGIS

El uso más común que se le ha dado a esta herramienta en este proyecto es cargar mapas de puntos en dos dimensiones. En QGIS existe el concepto abstracto de capa, el cual consiste en un conjunto de marcas geoespaciales dadas como tipo de dato geoespacial. En el caso de este trabajo práctico, nos basta con el tratamiento de capas de puntos.

Como ejemplo de carga de datos a continuación se procederá a cargar una capa a modo de ejemplo con multitud de datos en forma de puntos. Estos puntos se corresponden con los del fichero *PointsInterestCalifornia.csv*. A diferencia del fichero *PointsInterest.csv* usado en el caso práctico, este fichero contiene multitud de puntos de interés de toda California. Para cargarlos, se selecciona en la barra de herramientas de QGIS la opción *Capa→Añadir capa de texto delimitado…* Se nos abrirá un formulario de configuración como el de la Ilustración 44.

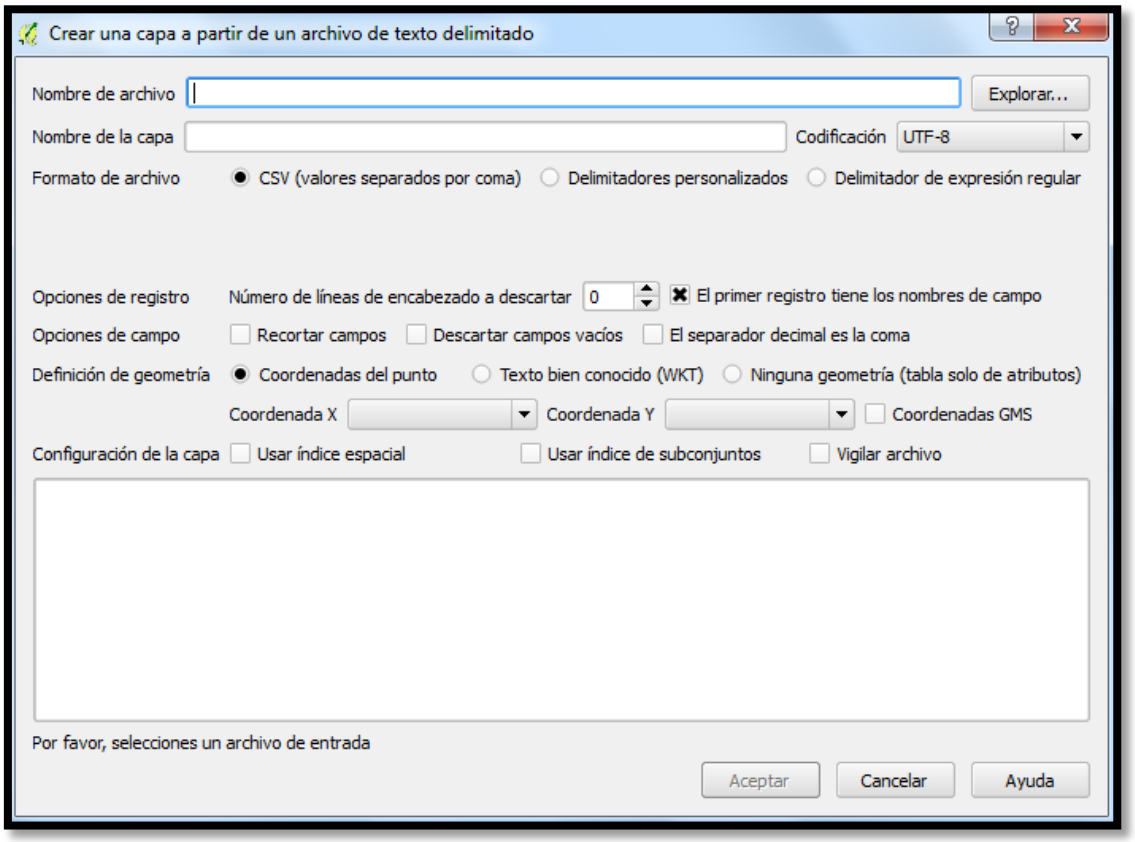

Ilustración 44: Formulario de creación de capa a partir de fichero de texto

En dicho formulario se ha de seleccionar el archivo, indicar un nombre a la capa y configurar las coordenadas X e Y. En el caso del fichero *PointsInterestCalifornia.csv* la longitud será la coordenada X y la latitud la coordenada Y, sin marcar la opción de coordenadas GMS. El resto de configuraciones se dejan como están. Al pulsar sobre *Aceptar* se validarán todos los puntos del fichero según las especificaciones que se han introducido, y si todos los puntos son correctos, se nos pasará a otra ventana donde se elige el sistema geoespacial que se desee para mostrar los datos. Se deja por defecto el que está escogido, WGS84, y se pulsa en *Aceptar*. A continuación se nos mostrará de forma visual cómo resulta nuestra capa de puntos. En el caso de *PointsInterestCalifornia*, la forma del conglomerado de puntos recuerda a la del Estado de California, sobre el cual están tomados los puntos. En la Ilustración 45 se puede apreciar la forma del dicho estado.

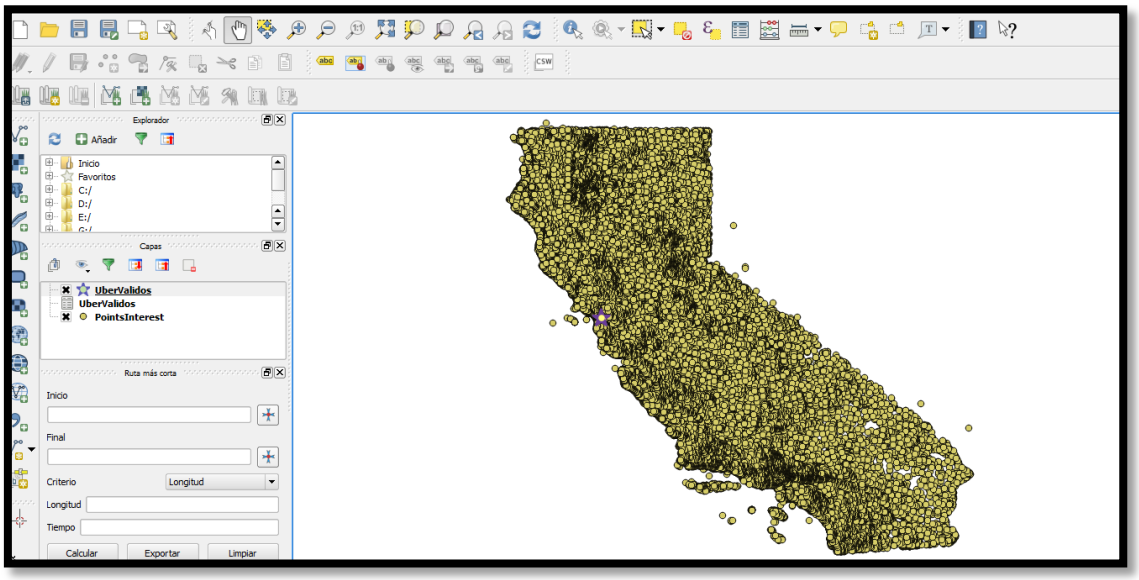

Ilustración 45: Representación de los puntos contenido en *PointsInterestCalifornia.csv*

Si se desea introducir otra capa encima de la ya creada se llevaría a cabo el mismo proceso de creación.

Anexo V: Scripts de configuración y creación de BBDD

Tabla 49: Anexo V: Creación de tablas

Creación de tablas
CREATE TABLE uber_temp (col_temp STRING); ***CREATE TABLE districts_temp (col_temp STRING);*** ***CREATE TABLE points_temp (col_temp STRING);***   ***CREATE TABLE Uber (idU INT, datetimeU STRING, latitudeU DOUBLE, longitudeU DOUBLE);*** ***CREATE TABLE Districts (nameD STRING, geometryD STRING);*** ***CREATE TABLE PointsInterest(namePI STRING, typePI STRING, latitudePI DOUBLE, longitudePI DOUBLE);***

Tabla 50: Anexo V: Copia de ficheros CSV en espacio HDFS

Copia de ficheros CSV en espacio HDFS
hadoop fs -copyFromLocal C:\Users\Administrator\Desktop\Uber.csv hdfs:///user/hadoop/ ***hadoop fs -copyFromLocal C:\Users\Administrator\Desktop\PointsInterest.csv hdfs:///user/hadoop/*** ***hadoop fs -copyFromLocal C:\Users\Administrator\Desktop\Districts.csv hdfs:///user/hadoop/***

Tabla 51: Anexo V: Carga de datos en tablas temporales

Carga de datos en tablas temporales
LOAD DATA INPATH 'hdfs:///user/hadoop/Uber.csv' OVERWRITE INTO TABLE uber_temp; ***LOAD DATA INPATH 'hdfs:///user/hadoop/PointsInterest.csv' OVERWRITE INTO TABLE points_temp;*** ***LOAD DATA INPATH 'hdfs:///user/hadoop/Districts.csv' OVERWRITE INTO TABLE districts_temp;***

Tabla 52: Anexo V: Inserción de datos desde tablas temporales

Inserción de datos desde tablas temporales
INSERT OVERWRITE TABLE Uber ***SELECT*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){1}', 1) idU,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){2}', 1) datetimeU,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){3}', 1) latitudeU,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){4}', 1) longitudeU*** ***FROM uber_temp;*** ***INSERT OVERWRITE TABLE Districts*** ***SELECT*** ***regexp_extract(col_temp, '^(?:([^_]*)_?){1}', 1) nameD,*** ***regexp_extract(col_temp, '^(?:([^_]*)_?){2}', 1) geometryD*** ***FROM districts_temp;*** ***INSERT OVERWRITE TABLE PointsInterest*** ***SELECT*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){1}', 1) namePI,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){2}', 1) typePI,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){3}', 1) latitudePI,*** ***regexp_extract(col_temp, '^(?:([^,]*)\,?){4}', 1) longitudePI*** ***FROM points_temp;***

Tabla 53: Anexo V: Adición de *Jars* de bibliotecas espaciales en Hive

Adición de Jars de bibliotecas espaciales en Hive
add jar C:\hdp\hadoop\hive-0.11.0.1.3.0.0-0380\lib\spatial-sdk-hive-1.0.1.jar; ***add jar C:\hdp\hadoop\hive-0.11.0.1.3.0.0-0380\lib\esri-geometry-api.jar;***

Tabla 54: Anexo V: Creación de funciones espaciales

Creación de funciones espaciales
create temporary function ST_Point as 'com.esri.hadoop.hive.ST_Point'; ***create temporary function ST_Contains as 'com.esri.hadoop.hive.ST_Contains';*** ***create temporary function ST_AsText as 'com.esri.hadoop.hive.ST_AsText';***

```
create temporary function ST_GeometryType as
'com.esri.hadoop.hive.ST_GeometryType';

create temporary function ST_Intersects as 'com.esri.hadoop.hive.ST_Intersects';

create temporary function ST_Equals as 'com.esri.hadoop.hive.ST_Equals';

create temporary function ST_GeomFromText as
'com.esri.hadoop.hive.ST_GeomFromText';

create temporary function ST_Area as 'com.esri.hadoop.hive.ST_Area';

create temporary function ST_Distance as 'com.esri.hadoop.hive.ST_Distance';

create temporary function ST_SetSRID as 'com.esri.hadoop.hive.ST_SetSRID';
```

Anexo VI: Acrónimos

ACID: *Atomicity, Consistency Isolation, Durability*. Término que expresa la función que las transacciones desarrollan en aplicaciones críticas para una misión. Acuñado por los pioneros en el procesamiento de transacciones.

APA: *American Psychological Association* (Asociación Estadounidense de Psicología). Organización científica y profesional de psicólogos estadounidenses fundada en 1892.

API: *Application Programming Interface* (Interfaz de Programación de Aplicaciones). Conjunto de funciones y procedimientos, los cuales abstraen un conjunto de operaciones y tratamiento de datos, con el fin de que otros programas hagan uso de ellas.

CSV: *Comma Separated Values* (Valores Separados por Comas). Nombre con el que se conoce a los conjuntos de valores separados por comas. Es común hablar de ficheros CSV, donde los valores que contienen van separados por coma, siendo idóneos para manejar información que se guarda en tablas.

ESRI: *Enviromental Systems Research Institute* (Instituto de Investigación de Sistemas Medioambientales). Empresa fundada en 1969 dedicada al desarrollo y comercialización de software para Sistemas de Información Geográfica (GIS). Su producto más conocido es ArcGIS.

GIS: *Geographic Information System* (Sistema de Información Geográfica). Herramienta o sistema mediante el cual puede estudiarse la información geográfica de conjuntos de datos. Puede referirse a programas como QGIS (*Anexo IV*) o a APIs que interpreten y operen con datos geográficos, como PostGIS.

HDFS: *Hadoop File System* (Sistema de Ficheros de Hadoop). Sistema de ficheros distribuido creado por Hadoop, el cual destaca por su baja tolerancia a fallos y la gran portabilidad que posee para instalarse en gran cantidad de máquinas.

HQL: *Hive Query Language* (Lenguaje de Consulta de Hive):

Proyecto GNU: *GNU is not Unix Project* (GNU no es Unix). Proyecto mediante el cual se diseñaron licencias contrarias al *copyright*, las conocidas como *copyleft*, a raíz de la proliferación del sistema operativo Unix y el interés de la comunidad de desarrolladores de poder moldearlo y editarlo.

SQL: *Standard Query Language* (Lenguaje Estándar de Consulta). Lenguaje de acceso a base de datos por excelencia, nacido junto con las bases de datos relacionales a principio de los años setenta. Existen numerosos lenguajes de acceso a datos que toman su base del SQL estándar, como MySQL o PL/SQL de Oracle. IBM definió el lenguaje como tal y Oracle más tarde lo incluyó en un producto comercial.

SRID: *Spatial Reference System Identifier* (Identificador de Referencia Espacial). Identificador único asociado con un sistema de coordenadas específico, tolerancia y resolución.

WGS84: *World Geodetic System 84* (Sistema Geodésico Mundial 1984). Sistema de coordenadas geográficas mundial que pretende representar cualquier punto dado de la tierra mediante tres puntos. Los cálculos se hacen sobre una forma perfecta de la Tierra, un elipsoide, llamado WGS 84.

WKT: *Well Known Text* (Texto Bien Conocido). Descripción estandarizada mediante caracteres ASCII de objetos espaciales, tales como puntos, polígonos, líneas, etc. Es un estándar promovido por la OGC (*Open Geospatial Consortium*).

Anexo VII: Definiciones

***Amazon's Dynamo*:** Base de datos NoSQL propiedad de Amazon que provee un servicio rápido, eficiente y escalable de almacenamiento de datos. De las tablas montadas en este sistema se puede extraer la cantidad de información que se desee, así como insertar del mismo modo. Dynamo se encarga de que estas operaciones masivas sean garantizadas mediante la ayuda de un número suficiente de servidores, que dinámicamente se unen para soportar la carga de trabajo o se desactivan según se necesite.

Argumento: En informática, campo de una expresión, método, función o comando. Se puede indicar por valor directo o como una referencia a un valor.

Atributo: Cada una de las propiedades de una entidad en base de datos. Cada columna de base de datos almacena un atributo distinto de la entidad.

***Big Table*:** Mecanismo no relacional de almacenamiento de datos distribuido y multidimensional basado en las tecnologías de almacenamiento propiedad de Google para la mayoría de aplicaciones en línea y back-end de la empresa / productos. Proporciona arquitectura de datos escalable para infraestructuras de bases de datos muy grandes. BigTable se utiliza principalmente en los productos de propiedad de Google, aunque algunos estén disponibles en aplicaciones de otros fabricantes de bases de datos.

Cláusula: Expresión utilizada en los lenguajes de consulta para indicar una condición sobre filas de la relación sobre la que actúan, normalmente condiciones de ordenación, agrupación y/o filtrado.

Cisco: Empresa estadounidense líder en fabricación, venta, mantenimiento y consultoría de equipos de telecomunicaciones, tales como *routers*, *hubs*, *switches* o dispositivos para redes de área de almacenamiento.

Comando: Expresión utilizada en los lenguajes de consulta para indicar la operación que se va a realizar sobre la tabla objetivo. Las operaciones más comunes son la inserción, la modificación, la selección y el borrado.

Consulta espacial: Consulta de bases de datos que contiene al menos una función espacial en su composición para lanzarla sobre datos espaciales. No confundir con consultas sobre datos espaciales, ya que en ese caso pueden o no llevar en su estructura funciones espaciales.

Creative Commons: Organización sin ánimo de lucro cuyo fin es proporcionar una serie de licencias flexibles a los creadores de contenidos. Esto quiere decir que el autor de un contenido puede cambiar a su gusto las condiciones de los derechos o licencias sobre su obra.

***Data set*:** En castellano: conjunto de datos. Término anglosajón utilizado para hacer referencia al conjunto de datos almacenados en una tabla de base de datos o matriz, y en general para cualquier grupo de datos agrupados en estructuras de datos.

Diagrama de Gantt: Herramienta que permite al usuario modelar la planificación de las tareas necesarias para la realización de un proyecto. Esta herramienta fue inventada por Henry L. Gantt en 1917.

Estilo APA: Manual de estilo de documentación y publicación, uno de cuyos aspectos son las citas y referencias bibliográficas. Es promovido por la *American Psychological Association* (APA), y es uno de los estándares más conocidos de referenciación bibliográfica.

Expresión regular: Secuencia de caracteres usada para identificar patrones dentro de una cadena de texto.

GeoMesa: Base de datos espaciotemporal construida sobre el repositorio basado en *column-family* Apache Accumulo, cuya principal función es el manejo de datos espaciales mediante consultas espaciales.

GitHub: Plataforma online de almacenamiento y distribución de proyectos software *open source*, cuyo principal fin es facilitar la compartición de código fuente entre desarrolladores.

Google Maps: Aplicación de Google Inc. que proporciona mapas de las regiones del mundo, pudiendo navegar por ellos y establecer rutas para seguirlas. Tiene varios niveles de detalle posibles, y según dónde se sitúe el usuario muestra los puntos de interés y las mejores formas de llegar al sitio que se le indique.

HBase: Sistema de almacenamiento masivo *open source* creado por Apache basado en la tecnología de almacenamiento distribuido *Big Table* de Google Inc.

Java: Plataforma de computación creada por Sun Microsystems. Está concebida a modo de máquina virtual con el objetivo de poder portar de un sistema operativo a otro el código compilado de las aplicaciones desarrolladas en la plataforma (*bytecode*). El lenguaje de programación más utilizado para crear aplicaciones en este entorno es el homónimo Java, pero cualquier lenguaje que genere un *bytecode* compatible es válido. Es una de las plataformas más extendidas para desarrollo y ejecución de aplicaciones.

MongoDB: Sistema *open source* de almacenamiento masivo NoSQL basado en documentos, desarrollado y mantenido por la compañía 10gen. Es uno de los puntales del movimiento *Big Data.*

Neo4j: Software libre de base de datos orientada a grafos implementado en Java. Es un motor de persistencia embebido, basado en disco, completamente transaccional que almacena datos estructurados en grafos.

***Open source*:** En castellano: código abierto. Término anglosajón que se utiliza para indicar que una licencia que aplica sobre cierto software es libre, por lo que el código de dicho programa puede ser accedido por terceros para entenderlo y/o modificarlo. En ocasiones se confunde con el término "gratis", ya que normalmente un software con este tipo de licencia puede usarse sin pagar un precio por él. Pero esto no es cierto, ya que por ejemplo, puede haber software gratis y

no disponer de licencia de código abierto, como por ejemplo las aplicaciones web Facebook o Twitter.

Python: Lenguaje de *scripting* independiente de plataforma y orientado a objetos, cuyo uso va desde la programación de páginas webs, pasando por la configuración de servidores y aplicaciones de Windows. Es uno de los lenguajes más utilizados en la actualidad, gracias en gran medida a su gran eficiencia y las numerosas plataformas en las que se puede desarrollar con él.

***Query*:** Término anglosajón que hace referencia a una consulta de base de datos. En el lenguaje castellano, y más concretamente en el argot informático, es bastante común su uso por ser más corta y sencilla de pronunciar que su homónimo "consulta".

Red Hat: Sistema operativo *open source* basado en Linux, creada y mantenida por Red Hat Inc. Es utilizado en gran medida para el uso y configuración de servidores, aunque en la actualidad Red Hat es también muy utilizado en entornos empresariales.

Space Base: Sistema de almacenamiento de datos espaciales. La característica principal de este sistema es que toda la información es almacenada en memoria principal, por lo que no usa disco para persistir la información. Esto hace que su uso esté dirigido a procesos que no estén enfocados a almacenar la información que generan a largo plazo.

***String*:** Tipo de dato muy común en multitud de lenguajes de programación, como Java o C#. Sirve para almacenar cadenas de texto, sea cual sea su longitud. Hive soporta el tipo *string* como tipo de dato en los atributos de las tablas.

Third-party: Expresión anglosajona para hacer referencia a lo que en castellano se conoce como "De terceros".

Tiempo real: Expresión que indica la inmediatez con la que se espera un resultado. En tecnologías de la información, este término recoge aquellas operaciones cuyos resultados deben de producirse al instante, como por ejemplo la barra dinámica de búsqueda de Google o la imagen proyectada en un televisor.

Transacción: Secuencia de operaciones realizadas como una única unidad de trabajo. Debe cumplir las propiedades ACID para ser considerada como tal.

Tupla: Tecnicismo informático con el que se conoce a cada una de las filas de una tabla en base de datos.

VirtualBox: Plataforma *open source* de virtualización de servicios tanto empresariales como para uso particular. Uno de los usos más interesantes de este servicio es la virtualización de sistemas operativos, lo que hace que se pueda trabajar sobre varios sistemas a la vez, además de hacerlos portables y dotarles de aislamiento frente al sistema anfitrión.

Windows Server: Línea de productos de *Microsoft Corporation*. Se trata de la versión de su sistema operativo orientado a servidores.